Analyzer

The analyzer is the third element of the chain (starting from the bottom). The task of the analyzer is to keep track of the analyzed files with the help of a tree: if a node is a directory then it has a list of children. This component will get from standard input a file or a directory and, while inserting all the files in the tree, it will assign them to managers in a fair way. The analyzer will also handle any changes in the number of the managers by spawning or killing some of them. The same goes for the workers amount which will be changed by sending this particular information to the managers. If needed the analyzer will send the list of children of a folder or, for the requested files, the sum of all the tables associated to the file, group by each character.

Structure

The structure of the analyzer is basically composed of five threads:

- readDirectivesLoop that reads directives from standard input
- fileManageLoop that handle the files and directories obtained from the directives
- sendFileLoop that comunicate with managers
- readFromFIFOLoop that reads the reporter's directives
- writeOnFIFOLoop that accomplishes reporter requests

Tree

All the infos about files and directories are stored in a tree; each node except the root node can have one parent and multiple children saved in a list. Root doesn't have a parent but has children, it also has the destructor that will be executed on each node when the function destroyTree is called. The basic TreeNode contains:

- data
- children
- parent

The data stored in the TreeNode is a FileInfo instance that consists of:

- name
- fileTable, table that contains all the occurrences of the characters found in a specific file
- path
- isDirectory, flag that tells if the current node is a file (value 0) or a directory (value 1)
- isRequested, flag that tells if the current node is requested (value 0) or if it is not (value -1)

File/Directory insertion When the analyzer receives a file or a directory to process it stores all the information in order to mantain a sort of file system

hierarchy. When a path is obtained the analyzer will firstly compact it by removing references to the same folder and then process the node where the insertion will begin, starting from root of the tree if the path is absolute or the current folder if the path is relative, and by going up the chain of fathers or the chain of children. Basically the insertion function will first of all check if the starting node has any children. If the List of children is empty it will attach the new node to the current one; otherwise, each child of the current node will be scanned and, if a current portion of the path of the file to insert is contained in a node (which is a directory) the function will change the current node with the matching one and will continue to analyze the children of that one; this process will end when the correct location of the file is found (i.e. the least common ancestor or the file itself if was already inserted).

Read Directives Thread

In the Read Directives thread all inputs from the stdin are analyzed and, if they match a specific pattern, (path, number of managers, number of workers) a list of operations will be performed in this order:

- if the number of mamagers changed then a certain amount of process will be spawned or killed based on the value stored in a shared variable
- the given path is checked in order to verify if it's a directory or a file: if that's the case the path is enqueued in the candidateNode List, otherwise the thread will wait for the next input

The special path "///" is used if the user would like to only change the number of workers of each managers without adding new files to analyze.

Manage File Thread

If a file was inserted in the candidateNode List it is then extracted in this particular thread (but only if the readFlag is different from SUCCESS to avoid possible overwrites). The node extracted is then inserted directly into the tree and into the fileToAssign List if it's a file, otherwise it'll be spawned a find process, using a bash call, in order to check all the directories and their files nested in the one passed as input. This operation is exectuted recursevly for each of the directories founded in this process. In order not to block the operations in the other thread only a file at time is read from the spawned process and, as said before, it won't be possible to extract another node while this operation is still running in order to avoid any kinds of overwrite. The same insertion process for a file is performed in the same way for each result returned from the find.

Send File Thread

In this thread two different operations are performed:

- In the first half each managers is extracted and its pipe is checked: if there's something to read then the path of a file is read from the pipe. The

path obtained is checked with all of the files in the filesInExecution List of the manager and in case of a match a flag is setted. After this operation 129 numbers are read from the pipe and if the file was found in the previous operation the file's table is updated. If the file is also requested, before updating the amount of a specific character the actual value is substracted from the requestedFilesTable and then the same table is updated with the new amount. At last the control word sent from the manager is checked: if it's equal to "done" then the file is removed from the filesInExecution List of the manager (if the file is requested then the sendChanges flag will be set), if the control word is "undo" then no operation will be performed (if the file is requested the same operation will be perfomed as described above), otherwise the user will be informed of a comunication error between manager and analyzer.

- In the second half if the filesToAssign List isn't empty then the manager with less files is extracted from the priorityQueue. After that the file is first sent and then it's inserted in the manager's fileInExecution List. If the manager's pipe is full (or if the informations' size is greater than the reamining space on the pipe) then the file won't be sent, it won't be extracted from the fileToAssign List and the thread won't be stucked on the write.

Read FIFO Thread

After checking if the "reporterToAnalyzer" file exists a FIFO is opened between the analyzer and the reporter in O_RDONLY mode in order to be able to read the operations sent from the reporter. There are three main operations that can be performed:

- with "dire" the path, the number of managers and the number of workers are popped from the List of read words and then an operation which is identical to the one described in the Read Directives Thread will be executed.
- with "//" the element in the requestedFiles List and the requestedFilesTable are resetted and, while there are still element in the read word List, the files with a matching path to the one popped are setted as requested and their table is summed with the requestedFilesTable. After this operation the flag sendChanges is setted.
- with "tree" the path to retrieve is popped from the read word List and it is then saved inside toRetrieve variable.

If the word read is different from the previous ones and also different from "" (empty string) then the word is enqueued into the read words List ontherwise the FIFO is closed.

Write FIFO Thread

In this thread the informations requested from the reporter are sent. First thing first if the file "analyzerToReporter" doesn't exists then it's created. After this operation a FIFO in O_WRONLY mode is opened. In the first half of the loop the toRetrieve variable is checked and if it's different from NULL then a file with a matching path to the one saved in the variable is searched inside the tree and the children of the retrieved node are sent to the reporter using the format "tree" "number of children" "fileName1" "fileName2" ... In the second half of the loop the sendChanges variable is checked and if it's setted then the word "tabl" followed by the requestedFilesTable (which means 129 numbers) are sent to the reporter.

Considerations

The analyzer and his working chain (managers and workers) performs wells, the threads that compose it allows it to both answer at user input and process the given files at the same time.

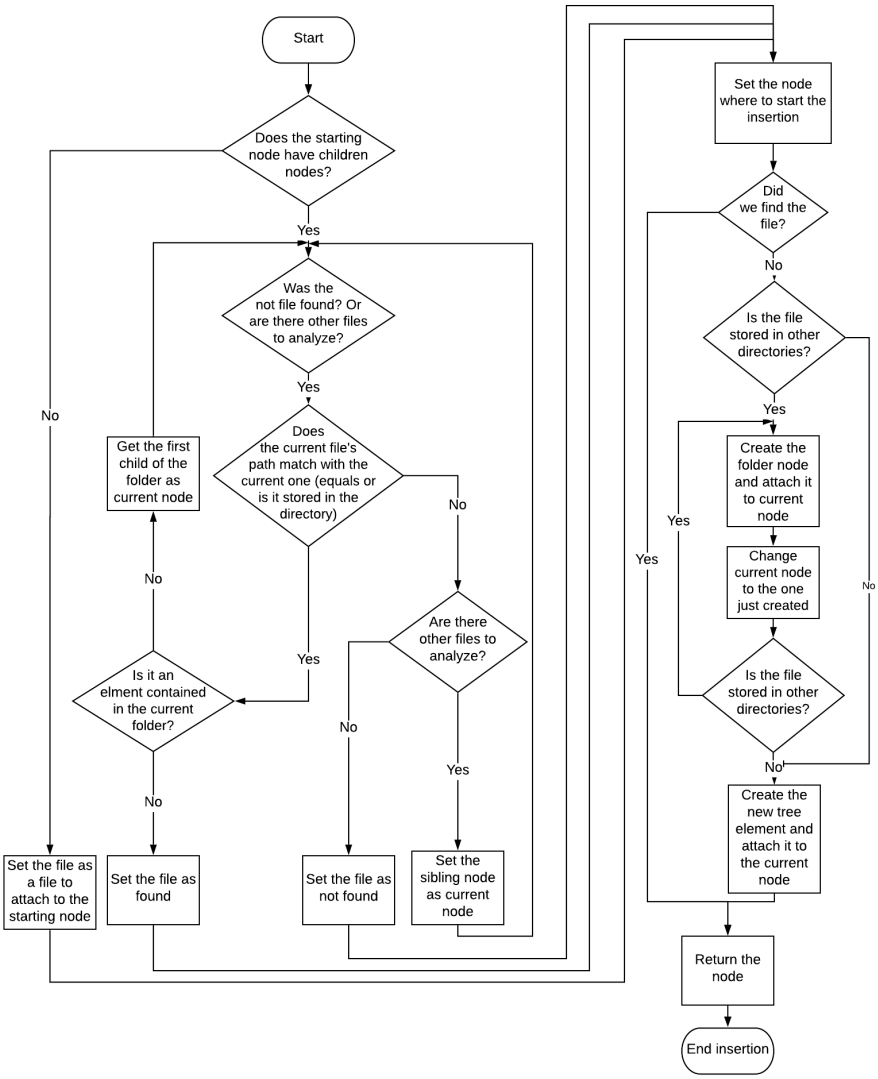

Figure 1: Node insertion

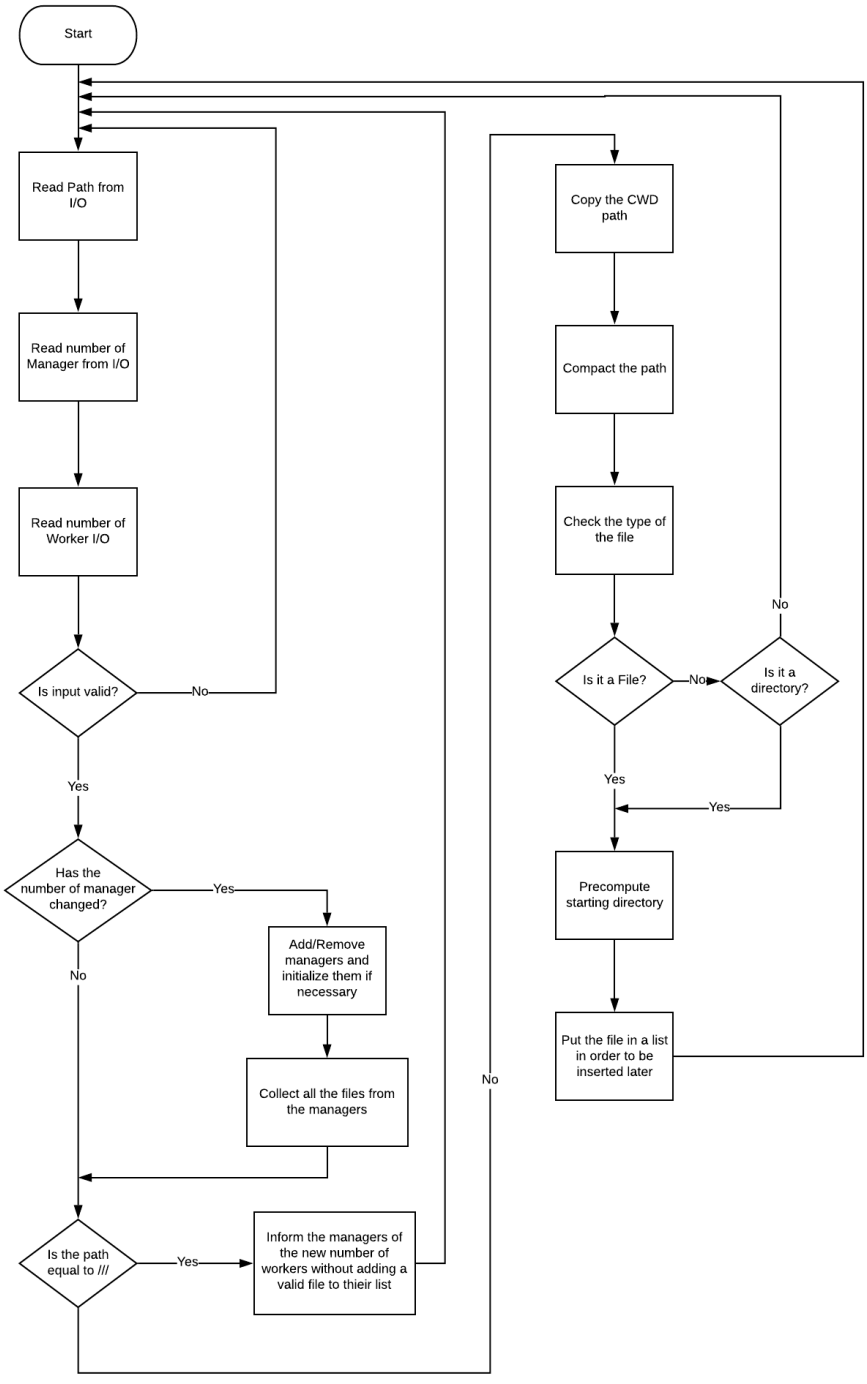

Figure 2: Read directives thread

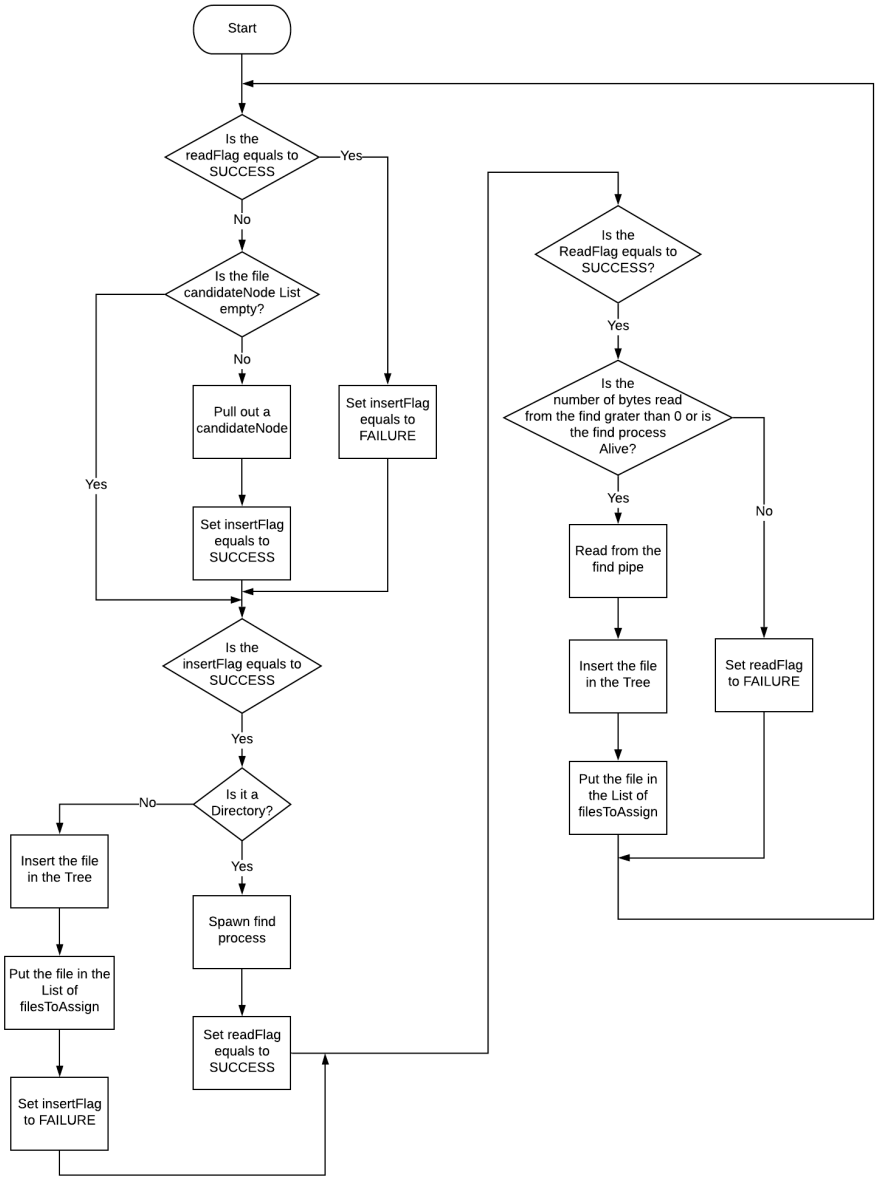

Figure 3: Manage file thread

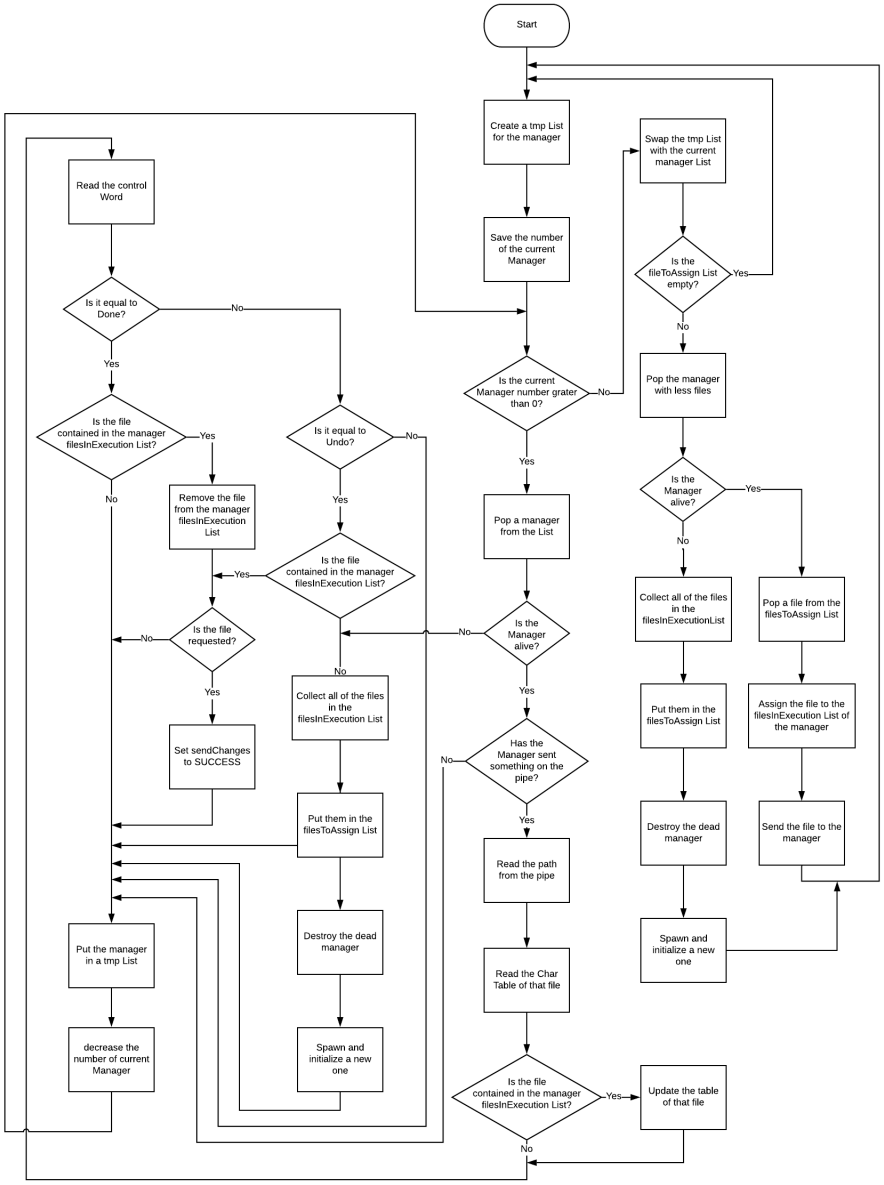

Figure 4: Send file thread

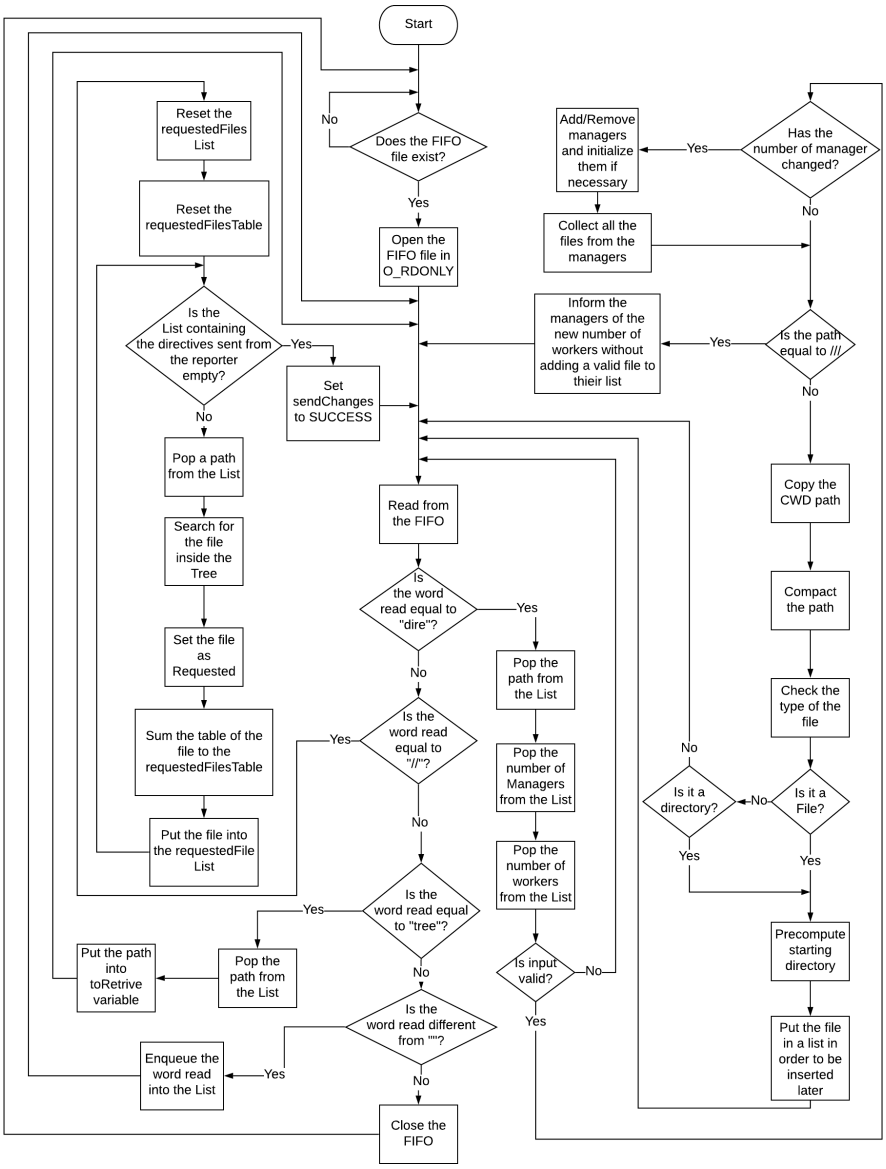

Figure 5: Read FIFO thread

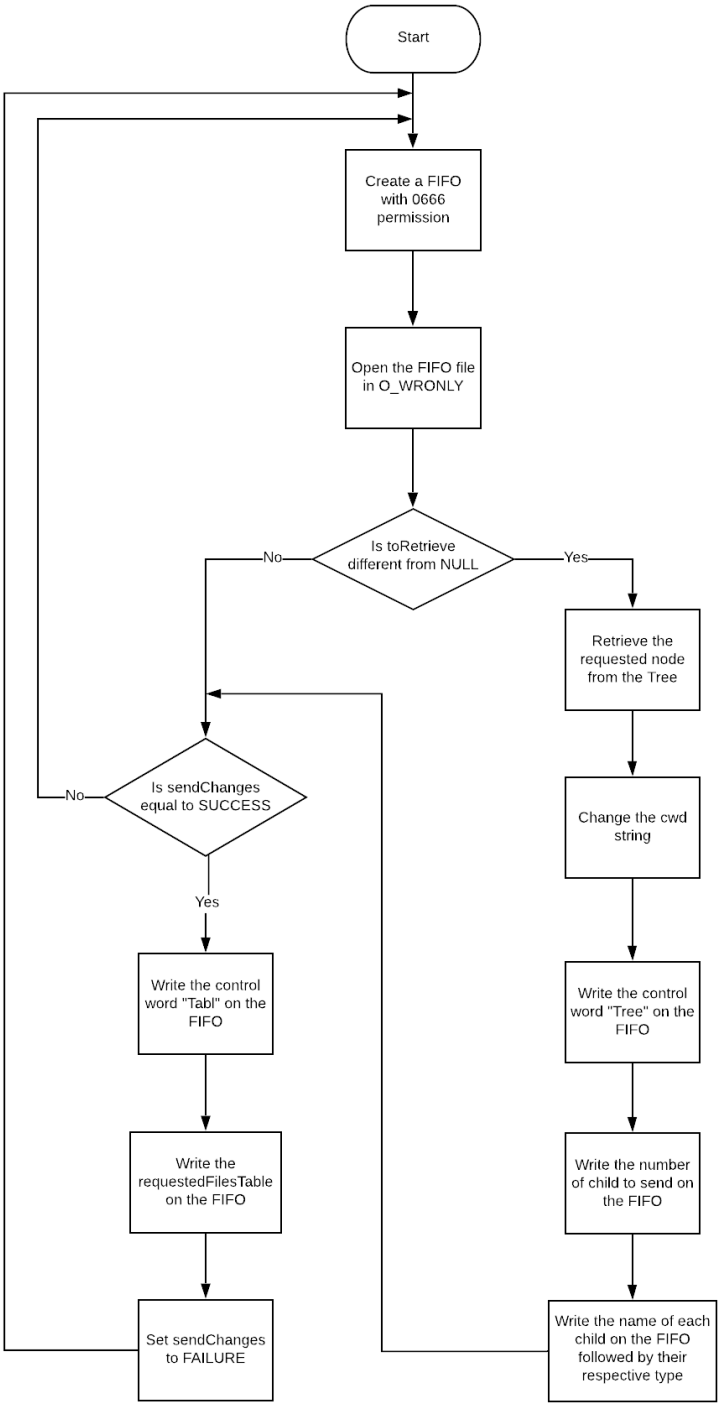

Figure 6: Write FIFO thread