Week 8: File & Threads Handling

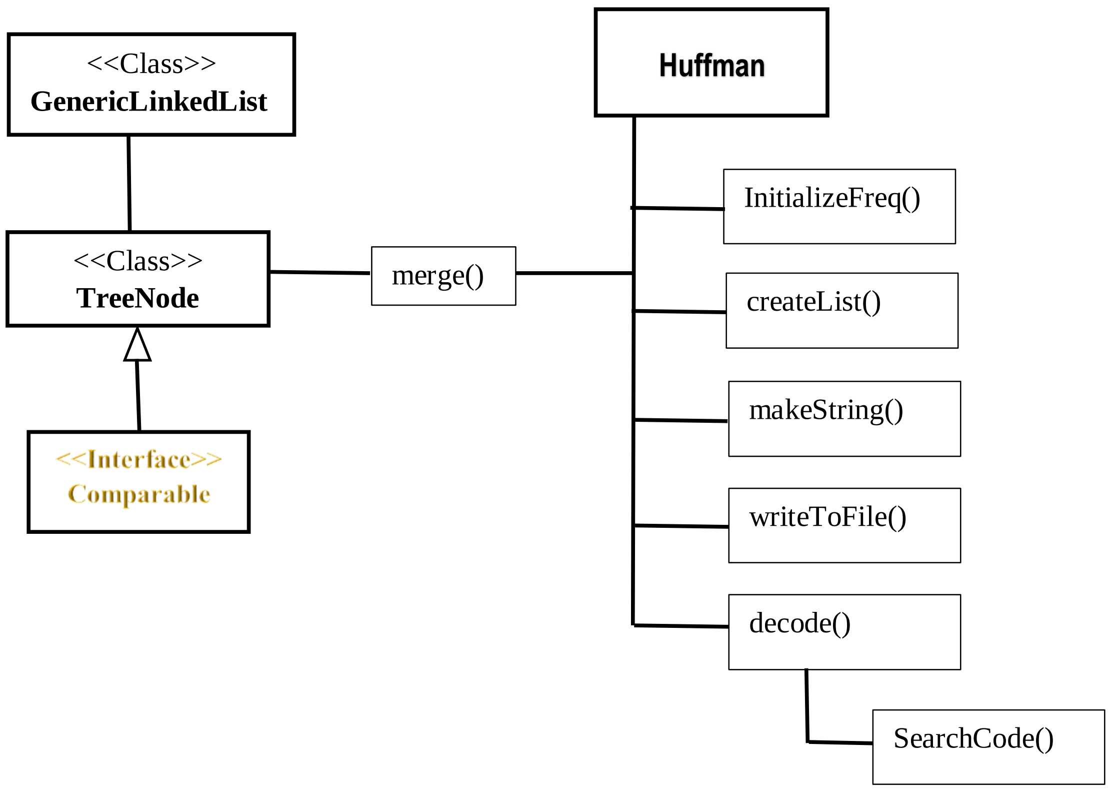
Assignment 1: Huffman Coding

In this week's assignment, we had to build on last weeks' problem on Huffman Coding. We had completed the process of encoding for it, and had to achieve decoding of the encoded text.

There were few problems which we ran into during the implementation of this code:

- For the assignment of Huffman, the previous week, we had done an 8-bit encoding of the individual text, but for this week's assignment we have used a 7-bit encoding, this is done because when we decode the 8-bit encoded text, we were getting some extra characters which we not the part of the actual text we had to achieve.
- There were a few letters which had their ascii values in the range of around 6K, so initially we had taken 400 as the array size, but had to increase it while implementing the decoding, because there was some loss of characters. Therefore, as per the suggestion we had from Professor we increased the size of the array considerably to accommodate the asci values. The only downfall of doing this upgrade was that the whole process of Huffman problem is taking a substantial amount of time to achieve the desired output.
- As mentioned earlier we are storing the text in a 7-bit encoding, so when we encounter a text line which is not a multiple of 7, we have leftover bits which when decoded gives extra text material for the extra bit space left in the last 7-bit chunk. For this we came up with a unique solution of appending the number of last bits left at the end of each encoded line. For example, if we have last 2 bits left, then we affix an integer 2 at the end of that particular line, this comes handy while decoding, since we know exactly ow many bits we need to assign for the last remaining chunk.

Basic work flow:



Assignment 2: Copying files

The assignment for this week asked us to simulate the functionality of the copy command. This process is done in two parts one using a single thread and the other using multi threads. Our implementation contains methods:

ParallelCopy*: Copies the contents of the files using multithreading*
SingleCopy*: Copies the files using a single thread*
CopyFile*: This method is shared by both the above methods to copy the folders' content.*
CheckConflict*: This method checks for any conflict in the destination directory – any name related conflicts. If the name provided by the user already exists in the drive, then this method appends a '_Copy' postfix to the desired folder name and then copies the files into it.*

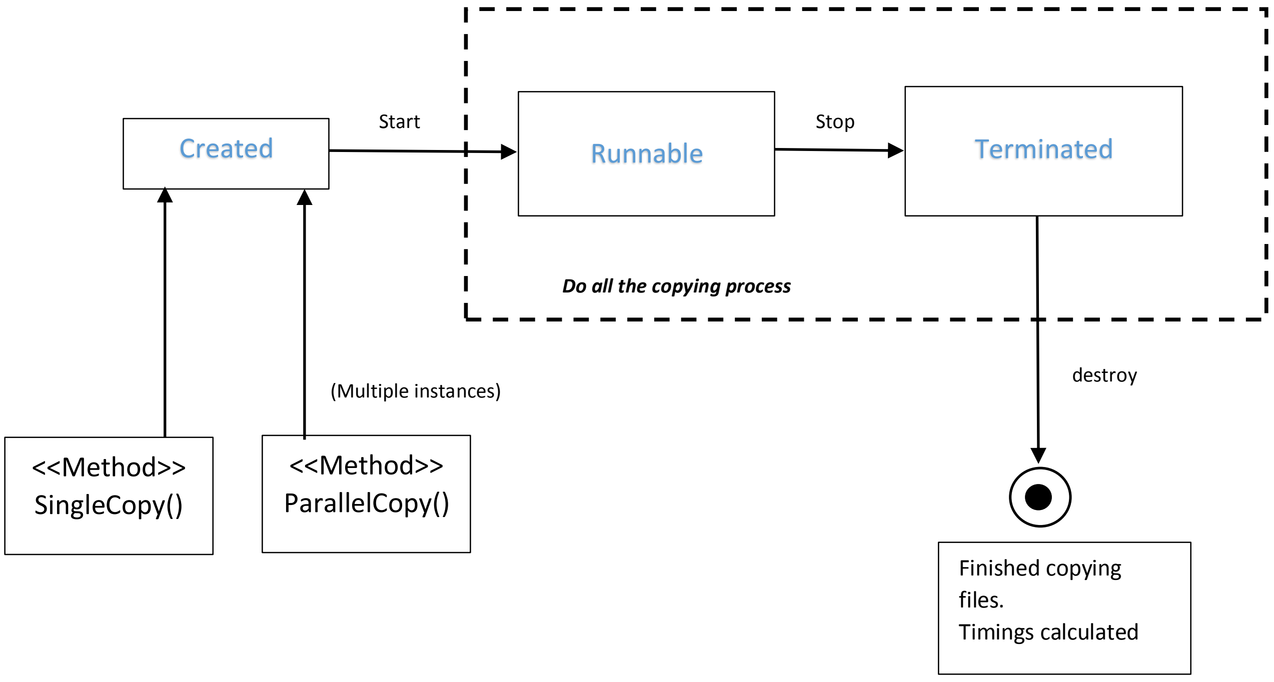
Single thread copying of files

In this method we have passed folder name, number of files in the that folder and the arguments taken from the user, such that the program understands where the copies of the files should be made and from where. We initially start a single thread which calls the SingleCopy function. This thread handles all the coping of the files inside the given folder. Additionally, we are also clocking the timings, checking how much time a single thread takes to copy all the files in the source file to the destination one.

Parallel copying of files

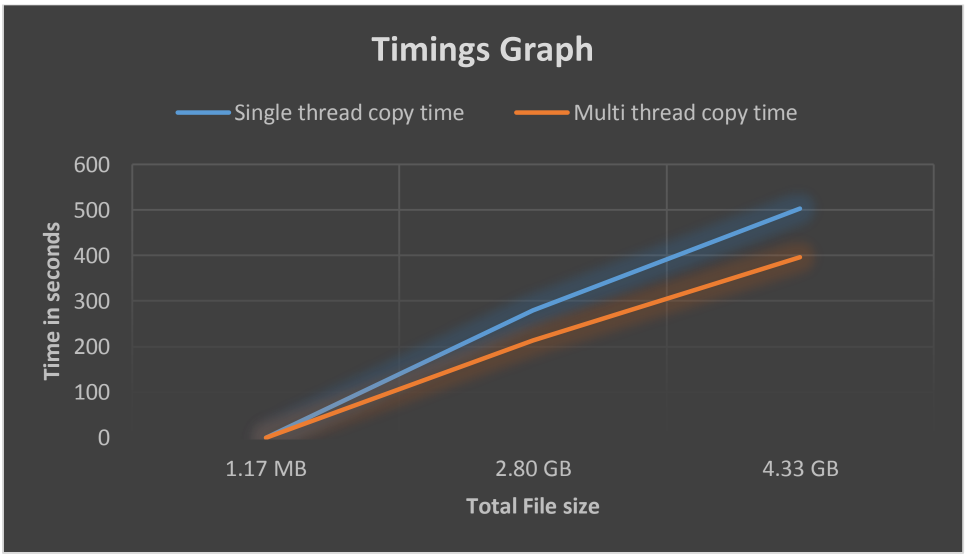
We have passed similar parameters in the method as in the single copy function, the major difference here is that each thread handles the copying of an individual file in the folder, so for example if we have 2 files in the folders then there will be two threads – namely Thread_1 & Thraed_2, started, where Thread_1 takes charge of copying file_1 and Thread_2 of file_2 or vice versa.

The code although being a really compulsive problem to solve, we surprisingly did not face any major problems while implementing it.



Timing reporting & Statistics:

Number of files	Total size	Single thread copy time	Multi thread copy time
10	1.17 MB	0.147	0.123
4	2.80 GB	278.9	214.11
4	4.33 GB	503.83	395.71



Conclusions:

In conclusion we can see from the timing reported that if we use multi-threaded program to copy multiple files at the same time, it clearly wins against using a single thread to copy the same amount of files. This proofs that work done parallelly is much more efficient that a single work pipeline.