Nishit Anand (na367)
Yash Phansalkar (yvp3)

Read Me

First of all, it was great working with the threads and multiprocessors. It is a great way to be time efficient and use multiple cores in order to pace up the process and improve the runtime of the program.

In this readme, we are going to define the way we implemented our programs and what difficulties did we face while writing our programs. Also, we divided the two parts in mutual understanding, by this I mean that one of us coded some part of threads and the other coded some parts of multiprocessors, then we exchanged thoughts how the program is suppose to work and we exchanged the programs to work on. By doing this, both of us were able to know how both programs work and it was great working together.

Below we are going to define the programs which we were suppose to implement and how their functions work.

Program: compressT_LOLS.c

In this program, we have a main function that takes in two arguments from the command line (excluding the file name). The first argument is the name of the text file that contains the string which we want to compress and the second argument is the divide the value (the number of splits we want to have the string compressed into).

Command Line Input:

```
./compressT_LOLS.C testplan.txt 3
<Name of Function> <Text File Name> <Number of Splits>
```

Main function does three processes. First, it open the text file and gets the length of the string via the use of fseeks and ftell. Second, the main method calculates the length of each string and then gets the mod of the length of string to add it to the first string. Third, it calculates the initial index to work on (basically the start and end index to work on based on length of each string), creates an array of thread, assign the structure to thread array I am using in the program and creates thread based on if it is the first string or not.

We have a compress function in our code that takes in a void structure pointer, then reads the entire data string from the text file provided to the program and copies the whole string to a new array. Then, we create a new temp array which is of the length of final index subtracted by initial index plus one (to add terminator). Further, the program checks if the character repeats int the string or not, runs till the final index of it, terminates it, and generates an output file.

Also, we join our threads in the main function and terminate the program, as it completes.

Program: compressR_LOLS.c

Unfortunately, we were unable to complete the multiprocessors portion, but we have a working child file. For our implementation we take two arguments in the parent file which are the filename of the file you want to compress and the number of processes you want to run. In our main file we then open the file you want to compress and use fseek to find the length of the string in the text file we want to compress. After this is completed, we figure out the length of the threads we are going to put into each new file for each process. In this, we find the length of the first string, as it will be the length divided by number of processes plus the remaining characters (mod value). After that we calculate the length of all the other strings that need to compressed in separate files. After which we go into a loop which is supposed to loop x amount of times x representing number of processes we take a moment to find the starting and ending indexes that we want to read from till. After which we use execlp to execute the processes at which time the worker file will come into play. Unfortunately, in using execlp that eliminates the loop and anything that happens after exec in the parent function. This was the issue we were dealing with prior to submission and that is why we don't have multiprocessors. The arguments passed through are the executable file compressR_worker_LOLS which should be compiled prior to running the parent function. The execlp will run the command for the worker file and have the remaining arguments of process number so is it the 1st or 2nd process running and then we have the start and end indexes we then pass those arguments into compressT_LOLS which will go into the file we want to compress and copy the chunk of the string we are compressing in said process. After which the string is passed to the textCompression text where the text is compressed to the format we want and then it is put into a new text file which is named the number of the process. As a result of the issue, we came across with execlp we are unable to continue the remaining processes but you can see the compression in process 1 is done correctly and is accurate. If you run the worker file alone with your manual inputs you can see that the worker file works properly and the reason this part of the assignment doesn't work is due to the execlp issue stated above. This algorithm works for all character types.

We do not have the file named timetest.pdf because our compressR_LOL.c program does not work, due to which we could not test both programs.