Name: Viraj Patel, NetID: vjp60

Name: Deepkumar Patel, NetID: dgp52

Procs Vs Thread

Part 1: Thread: How it's compiled and executed

- Step 1: gcc -Wall -pthread compressT_LOLS.c -o compressT_LOLS
- Step 2: ./compressT_LOLS fileToTest.txt 2
- Note: The last argument or the number of parts can be any number.

Part 2: Process How it's compiled and executed

- Step 1: gcc -Wall compressR_worker_LOLS.c -o compressR_worker_LOLS
- Step 2: gcc -Wall compressR_LOLS.c -o compressR_LOLS
- Step 3: ./compressR_LOLS fileToTest.txt 1
- **Note: It is very important to name the output file to "compressR_worker_LOLS" in step 1, because we use the name of the worker compiled file in our parent file to later call the worker file and pass all arguments using execl() function.**

Note: If the extension of the input file is not ".txt", it slightly affects the output file. It omits only one instance of the last character. For example: the ending "rrrr" will result in "3r", it won't read the last character "r". However, it works perfectly fine with files that have ".txt" extension at the end. In this case, the above example prints "4r", upon using .txt extension.

How our program takes inputs?

Our program takes two arguments mainly the uncompressed file name along with ".txt" extension and number of parts/threads/processes. If the argument is not equal to three then, it shows an input error. If the file is null then it shows an error. Otherwise, the program proceeds further and get the number of character in the file and try to divide equally among the number of parts/threads/processes. Our program also checks to see if the last argument is a digit and that total number of characters is greater or equal to number of thread.

How our program divides the file to parts?

First the program gets the number of parts/threads/processes, and then it creates an array of that many parts for later access. After that it gets modulo between the total character and number of parts. In addition, it also divides the total character and number of parts and gets the remainder. If the modulo is not even, then it means that the file is divided unevenly. At this point the program will have the number of characters each thread/process will compress, and using those values we can compute the stating index of each thread/processes.

File divide in equal parts: The array that we created earlier will be used to calculate the number of characters each thread will do. First, set the array values to zero. After that using modulo we increment the zero to one. For example, if the modulo is 2, and the array A, then A[0] = 1, and a[1] = 1. Adding the entire array with the remainder will give us the final result that is evenly as possible. The threads/processes that spawn first will do more work than later threads and processes.

File divide in unequal parts: It goes through the array and sets its value to result of total character by total parts.

Compression Algorithm: How it works?

Our compression function takes in five parameters. The name of the file, basically we will run our compression function on this file. Total character: is the number of character this thread/process will compress. Starting index: this is the index at which the thread/process will start. Thread number: is used to format the output file, for example, 0,1..n at the end of the file name. Parts: to check if it's one process or one thread, then output the file name without appending anything to the output filename.

Our algorithm uses for loop to go through the number of character it is provided. It will skip any non-alphabetic characters, which include space, newline or any symbol. Using a variable and a counter we keep track of the character and the number of time it has appeared, if the new character is different from what's in the variable then we output the variable and counter to output file. Otherwise we keep going through the loop until it ends.

Does the program overwrite?

If the output file already exists, and it has the same name, then the program overwrites the old files with the new file. It will not ask user for any confirmation.

Parent process/thread: What I do?

- Divide the file into even parts
- Give child/worker their starting index

Child/worker process/thread: What I do?

- Create output file along with correct filename
- Run compression function on the file