Perfect Numbers by Multithreading

Aaryan, CO21BTECH11001

Contents of the zip file:

Input file: input.txt contains the inputs to be given to the program.
Report: Assign2Report_CO21BTECH11001.pdf
Source code: Assign2Src_CO21BTECH11001.c
Readme file: Assign2Readme_CO21BTECH11001.txt

Working of code:

It can be described as follows:

1. To store the perfect numbers identified by the threads, linked list is used.

2. Function check_perfect takes an integer as input and returns true if the number is perfect. Otherwise false.
   For a number n, the time complexity of check_perfect is $O(n\sqrt{n})$.

3. The input file is given as command line argument.
   E.g., ./a.out input.txt
   Program will read input from the file using fscanf.

4. For every input n and k, it makes a directory named “Input_ct” where ct is the input number.
   For e.g., the directory of the first input will be named “Input_1”.

5. “main_file” is a pointer to log file generated by the main thread.

6. An array “files” of k file pointers is made.
   files[i] is a pointer to log file, which will be generated by ith thread.

7. An array “arrayOfPerfectNumbers” is made, which contains the head nodes of k linked lists.
   If a number “num” is identified as a perfect number by the ith thread, it

will be inserted in ith linked list, i.e., arrayOfPerfectNumbers[i].

8. Then, k threads are created. Each thread is responsible for a particular set of numbers.
   First (k-1) threads will be assigned (n/k) numbers each.
   Rest of the numbers will be assigned to kth thread.

   The schematic distribution of numbers to threads is done as follows:

   1st thread: 1, n, 2, n-1, 3, n-2…., i, n-i+1

   2nd thread: i+1, n-i, i+2, n-i-1, …, j, n-j+1
   .
   .
   .
   .

   (k-1)th thread: o, n-o+1, o+1, n-o, …, p, n-p+1

   (k)th thread: p+1, n-p, …, q, q+1

9. Reason behind this distribution is:
   Since check_perfect takes $O(n\sqrt{n})$ time to check if a number is perfect or not, each thread should be given big numbers as well as small numbers so that there is a fair distribution of operations performed by each thread.

10. This distribution is done by creating two parameters:
    int num_from_start;
    int num_from_end;

    num_from_start is initialized with 1, and num_from_end is initialized with n.

    Each thread will check num_from_start and num_from_end alternatively and increment and decrement them, respectively.
    This way, all numbers from 1 to n will be checked by all threads, and no

two threads will be checking the same number.

11. For passing all parameters to a thread, a struct data is created, which contains the following fields:
    a. int *start: Number from the beginning where the thread will start processing
    b. int *end: Number from the end where the thread will start processing
    c. int max_numbers: Maximum numbers to be processed by the thread
    d. bool last: True if the thread is last thread, false otherwise
    e. FILE** file: Log file created by thread
    f. struct node** head: Head of list where thread will store the numbers which are identified as perfect numbers.

12. Threads will be executed in the runner function.
    Let the number "num" is checked by a thread.
    a. If the number turns out to be perfect: It will add the line "num: Is a perfect number" to the log file of this thread by using the file parameter.
       Also, it will insert this number in the linked list (whose head node is the parameter head) by using the Insert function.

    b. If the number is not perfect: It will add the line "num: Not a perfect number" to the log file of this thread by using the file parameter.

13. After all threads are executed, the main thread will traverse through "arrayOfPerfectNumbers" and add these lines to log file of main thread:
    Thread1: num1 num2 num3 ...
    Thread2: num5 num6 ...
    I.e., it will insert all numbers identified as perfect numbers in ith line of log file generated by the main thread.

14. After the successful execution of all threads, it will print a completion message in stdout stream and deallocate the memory allocated in

heap, and close file pointers.

15. Each directory will consist of the following files:
    Main.log: log file generated by main thread
    Thread_1.log: log file generated by 1st thread
    Thread_2.log: log file generated by 2nd thread
    .
    .
    Thread_k.log: log file generated by kth thread