Programming Assignment 2: Finding Perfect Number

CO21BTECH11004

A global array is initialized ‘int* arr;’. It will be updated as the program runs and at the end will contain the data which number is perfect or not from 1 to n.

Function **IsPerfectNumber(int n)** checks whether a number is perfect or not. If it is a perfect number then IsPerfectNumber return **1** else return **0**.

Function **ThreadFunc(int mod),** is called by a thread. It first gets *n* and *k* from the global array. Then checks on the number set of a particular thread whether the numbers are perfect or not. If it is a perfect number then write in the thread output file **"Is a perfect number"** and **update** the **value at that index** in global **array to 1** else write in the output file **"Not a perfect number"**. This set of numbers for each thread is generated by the remainder number give when divided by k. Here function parameter ‘mod’ decides, the number from 1 to n which gives remainder ‘mod’ when divided by k will be operated by the thread which gives argument ‘mod; to ThreadFunc.

In the main function, *n* and *k* are read from the input file **(“input.txt”)**.

The global array is allocated (n+2) blocks of integers. First two element of array contains n and k (arr[0] = n, arr[1] = k). Then n blocks are initialized to 0.
The $(i + 2)^{th}$ index in the array represents $(i + 1)^{th}$ number and its value represents whether i+1 is perfect or not (i from 2 to n).
For eg:- arr[0+2] = 0 , tells that 0+1 = 1 :- 1 is not a perfect number.
arr[5+2] = 1, tells the 5+1 = 6 :- 6 is a perfect number

K threads are created using pthread_create(...).
The number from 1 to n are divided to k threads by the following distribution:-
Thread0 :- k,2k,3k,... (for i from 1 to n :- (i%k == 0))

Thread1 :- 1,k+1,2k+1,... (for i from 1 to n :- (i%k == 1))

Hence for i^{th} thread, it has to check on the number from 1 to n which give remainder *i* when divided by k. (*i* from 0 to k , excluding k)

Each thread process runs ThreadFunc with parameter 'mod', whose function is described earlier.

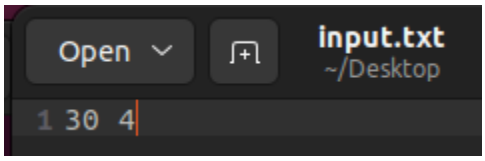
After all the threads exits, the parent process runs a loop k times executing pthread_join.

Then the parent process creates its output file and write number for which array value is 1 under their corresponding process , it finds corresponding process by checking the mod by k of number.

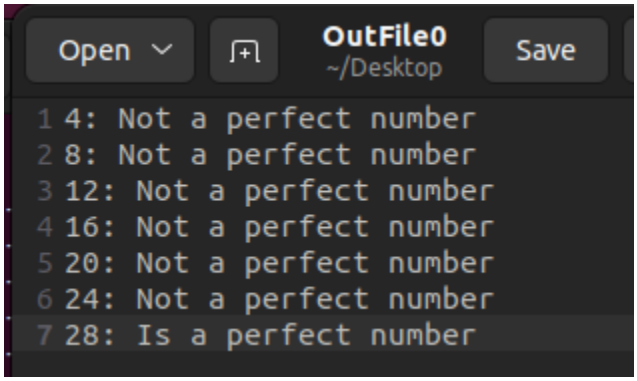
Close the input and output file.
Free the memory allocated to the global array.

For n=30, k=4. Illustration of code is demonstrated.

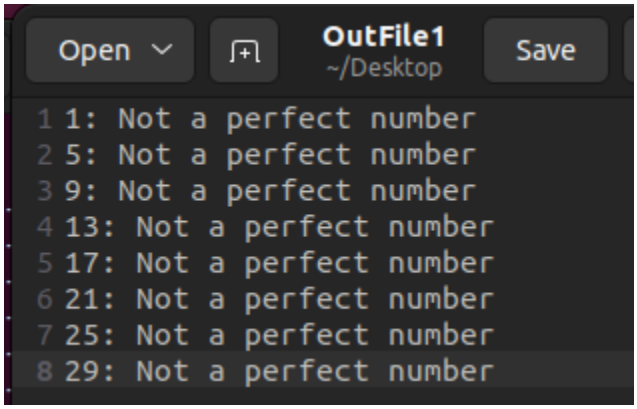
Input file:-



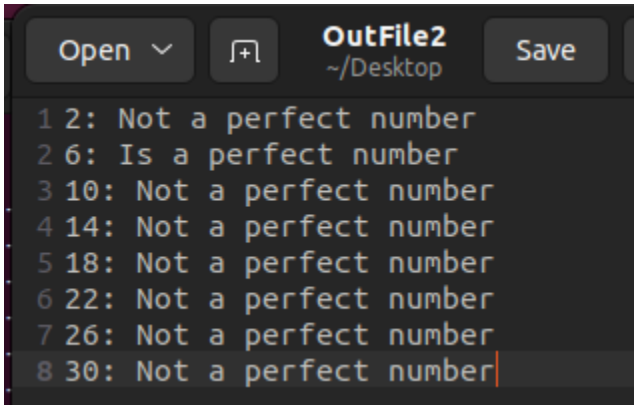
K Thread Output Files:- 4 Output File (OutFile0, OutFile1, OutFile2, OutFile3)
OutFile0.txt :-



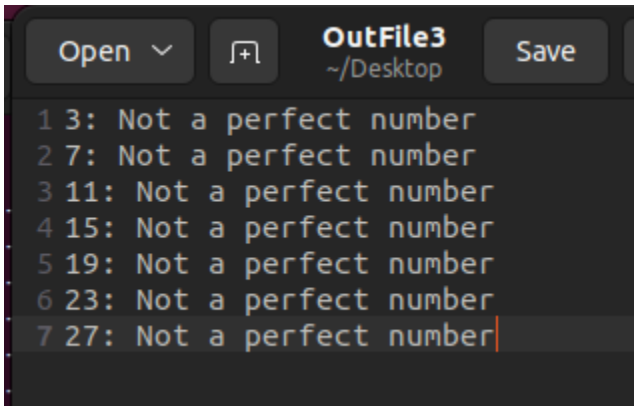
OutFile1.txt :-



OutFile2.txt :-

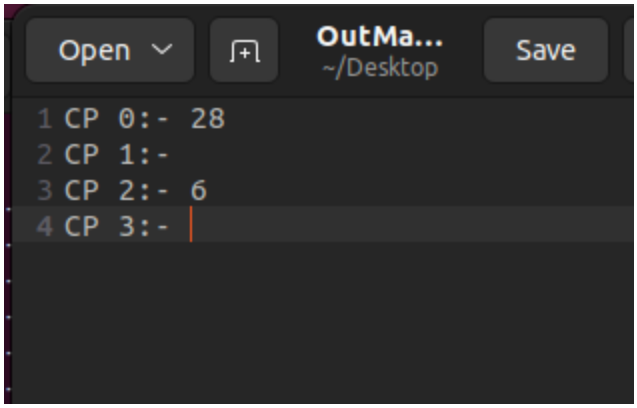


OutFile3.txt :-



Parent Output File:-

OutMain.txt :-



Initially Global Array (from index 2 to n+1) :- // Numbers 1 to n

```
1 :- 0 ; 2 :- 0 ; 3 :- 0 ; 4 :- 0 ; 5 :- 0 ; 6 :- 0 ; 7 :- 0 ; 8 :- 0 ; 9 :- 0 ;
 10 :- 0 ; 11 :- 0 ; 12 :- 0 ; 13 :- 0 ; 14 :- 0 ; 15 :- 0 ; 16 :- 0 ; 17 :- 0 ;
 18 :- 0 ; 19 :- 0 ; 20 :- 0 ; 21 :- 0 ; 22 :- 0 ; 23 :- 0 ; 24 :- 0 ; 25 :- 0 ;
 26 :- 0 ; 27 :- 0 ; 28 :- 0 ; 29 :- 0 ; 30 :- 0 ;
```

Global after k thread execution is done (from index 2 to n+1):- // Numbers 1 to n

```
1 :- 0 ; 2 :- 0 ; 3 :- 0 ; 4 :- 0 ; 5 :- 0 ; 6 :- 1 ; 7 :- 0 ; 8 :- 0 ; 9 :- 0 ;
 10 :- 0 ; 11 :- 0 ; 12 :- 0 ; 13 :- 0 ; 14 :- 0 ; 15 :- 0 ; 16 :- 0 ; 17 :- 0 ;
 18 :- 0 ; 19 :- 0 ; 20 :- 0 ; 21 :- 0 ; 22 :- 0 ; 23 :- 0 ; 24 :- 0 ; 25 :- 0 ;
 26 :- 0 ; 27 :- 0 ; 28 :- 1 ; 29 :- 0 ; 30 :- 0 ;
```