OS ASSIGNMENT-1 REPORT

Rajiv Shailesh Chitale
CS21BTECH11051

Input:
A file name is taken as a command line argument or input. The file contains parameters N and K. The program finds perfect numbers from 1 to N using K processes.

Forking:
A for loop runs with k iterations. A fork from the main process takes place every iteration. Each iteration, the child process with pid==0 breaks from the loop. The counter of the loop at that stage is used as its identifier, setid.

The parent (pid>0) calls the function manage() whose purpose is to distribute the numbers among the children and later consolidate the results. The children(pid==0) call function solve() to check a certain set for perfect numbers.

Shared Memory:
A shared memory is created between the parent process and each child process using shm_open(). It is allocated 64*sizeof(long long) = 512 bytes, using ftruncate().

Two buffers are created in each shared memory:

- tasks - parent is producer, child is consumer; can store 15 long long ints
- results - child is producer, parent is consumer; can store 39 long long ints

Due to the requirement of two way communication, both child and parent are are given read and write permissions with the help of PROT_READ | PROT_WRITE flags in mmap.

Buffers:
The details for a buffer are stored in a struct sharedBuffer. It has attributes which contain:

- shmname //Name of shared memory, based on setid
- statptr //Pointer to common variables (ints such as in, out, complete_flag)
- bufptr //Pointer to data of the buffer (elements are long long integers)
- LEN //Length of the buffer

The buffers are implemented using circular queues. The producer enqueues the data bufptr[in] and consumer dequeues the data bufptr[out], incrementing the values of in and out respectively in the shared memory.

If there is nothing to read, the child waits in a loop. If the buffer is full, the parent does not wait in a loop, but instead switches its attention to the next child. This is discussed in the next section.

Data Distribution:
The data is not distributed with a formula beforehand, but dynamically during runtime with the help of the shared memory buffer. The parent is in a circular loop over the child processes. It tries to enqueue one value, and moves on to the next child. If the buffer has space, the child will accept the new data to work on. If the buffer is full, the new data will not be enqueued. This automatically balances load between busy and free processes.

Checking Perfect Numbers:
A number is perfect if the sum of its factors excluding the number itself is equal to the number. Let us consider the number n. Suppose a is a factor of n, then there is a corresponding factor b such b = n/a. Further, for every b>sqrt(n) we obtain a< sqrt(n).

The code uses this knowledge to narrow the search for factors down to floor(sqrt(n)). The other factors are obtained as b = n/a. Each such pair of factors is added to a partial sum (which is initialized with 1). The case b=a=sqrt(n) is handled separately to avoid repetition of factors. If the partial sum exceeds the number, the number is immediately deemed to not be perfect (return value 0). Finally, if the sum equals the number, then isPerfect() returns the value 1.

Numbers which are found to be perfect are stored in a local array in each child.

Child termination and wait:
Once all numbers till n are exhausted the parent sets a flag. The child processes will not standby for new data, and are able to terminate. Before they terminate, they copy their results from the local array to the shared buffer- results. The parent waits for all the child processes to complete by looping wait() until it fails (ie. returns -1).

Output:
The parent loops through each child, and reads from each result buffer. It combines all of these results into one text file, OutMain. The perfect numbers are grouped by the process that found them. Additionally, every child process has a log file, OutFileX. It writes the result of every number it checks into this file using fprintf().

Analysis:

Outfile for N=50000, K=4:

```
P1: 6
P3: 28 496 8128
```

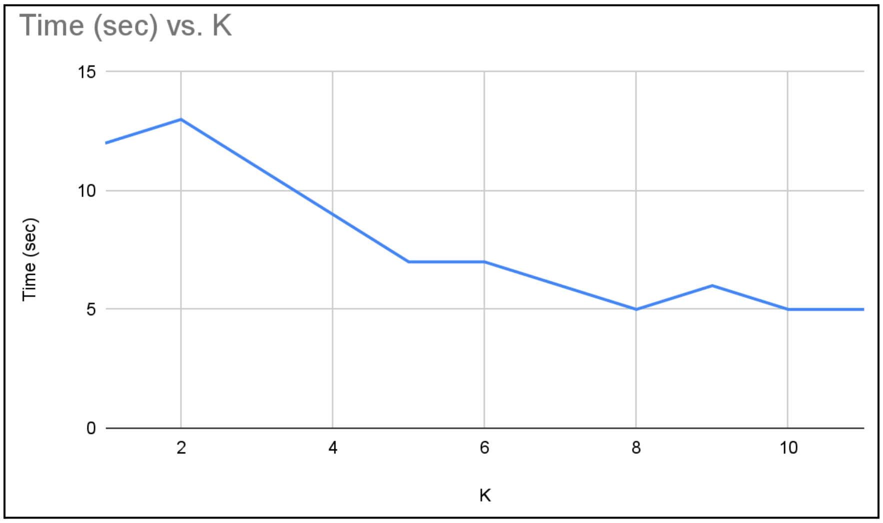
There are no new perfect numbers even for N=2000000, K=7:

```
P3: 28
P5: 6
P7: 496 8128
```

Thus the occurrence of perfect numbers is very rare. These four values also happen to be even.

Time to the nearest second for N=2000000,

K	Time (sec)	K	Time (sec)
1	12	6	7
2	13	7	6
3	11	8	5
4	9	9	6
5	7	10	5



Analysis:

- More processes running concurrently makes the computation faster.
- My computer has 4 cores with 2 threads each, which equates to 8 CPUs. Running more than 8 processes concurrently is not possible with my hardware, thus there is no benefit beyond K = 8 on my hardware.
- Two processes are not productive enough to overcome the inefficiency caused by the parent process having to switch its focus between children