Operating Systems II
Spring 2024
Lab Exam

Soham Rajesh Pawar
CS22BTECH11055

April 30, 2024

1 Coding Approach

The code uses multithreading to get the perfect numbers within a specified range.
Perfect numbers are numbers whose divisors(excluding itself) sum up to the number itself. Eg. 6, 28, 8128, etc.

1.1 Common Routine:

Listing 1: Routine 1

```
void routine(int index)
{
    for (int i = index; i <= n; i += k)
    {
        if (perfect(i))
        {
            sem_wait(&print);
            outfile << i <<" : Found by thread " << index << "\n";
            sem_post(&print);
            total.fetch_add(1);
        }
    }
}
```

1. **Note that the code contains an atomic int variable `total` i.e responsible for counting the number of perfect numbers encountered.**
2. **Each thread processes the numbers that are i modulo k, where i is the index/id of the thread. Note that thread k processes numbers that are 0 modulo k**
3. **Each thread makes a call for the `perfect function` which tells them if the number they are currently processing is perfect or not.**
4. **If yes, then they print their discovery in outFile.txt atomically(to avoid messy printing) and also increment total. Else move on.**

1.2 Function:

Listing 2: Routine 1

```
bool perfect(int n)
{
    int sum = 0;
    for (int i = 1; i < n; i++)
    {
        if (n % i == 0)
            sum += i;
    }

    if (sum == n)
        return true;
    else
        return false;
}
```

The correctness of the function is obvious.

2 Verification

1. **Verification can be done by referring to sources which have already performed the said task successfully.**

3 Output Time Analysis

3.1 Total Time vs N:

`N is in powers of 10, K is 8`

N	Time (seconds)
3	0.001484
4	0.029145
5	2.74737
6	104.43

Table 1: Execution Times

3.2 Total Time vs Number of Threads, K:

`N is 10000`

K	Time (microseconds)
1	0.220093
2	0.109615
4	0.055748
8	0.028887
16	0.034837

Table 2: Execution Times