Assignment 1 Report

Tanmay Goyal
AI20BTECH11021

Program Design:

The program begins with reading from the input file ‘input.txt’, and getting the values of N, the number till which we must check, and K, the number of processes we wish to create. If the file cannot be opened and/or if N or K is less than or equal to zero, we return an error.

The program has been designed in a way that all process has a uniform load. This has been done by ensuring each process gets it’s own share of larger and smaller numbers. For example, if we must divide 97 amongst 10 processes, the following is the allocation:

Process1	1	11	21	31	41	51	61	71	81	91
Process2	2	12	22	32	42	52	62	72	82	92
Process3	3	13	23	33	43	53	63	73	83	93
Process4	4	14	24	34	44	54	64	74	84	94
Process5	5	15	25	35	45	55	65	75	85	95
Process6	6	16	26	36	46	56	66	76	86	96
Process7	7	17	27	37	47	57	67	77	87	97
Process8	8	18	28	38	48	58	68	78	88	-1
Process9	9	19	29	39	49	59	69	79	89	-1
Process10	10	20	30	40	50	60	70	80	90	-1

Here, -1 indicates that we have exhausted all the numbers, and that there is no available number for the given slot.

We then create two different shared memories with the names ‘SharedMemory1’ and ‘SharedMemory2’, each having their own pointers ‘ptr1’ and ‘ptr2’. SharedMemory1 is to keep track of the

pairs of perfect numbers and the process that identified the same, and SharedMemory2 is to keep track of the offset for the pointer. An example would be:

Shared Memory1 ->

Pointer	0	1	2	3	4	5		
Value	6	ProcessX	28	ProcessY	496	ProcessZ		

SharedMemory2 –> 6

We then fork, creating child processes. The child process retrieves the number allotted to them, checks if it is not -1, and then checks if it's a perfect number or not. It then accordingly writes to the file they open. In case it is a perfect number, the child process also adds the number, and its own process number to SharedMemory1, and then increments the offset in Sharedmemory2 by 2. It uses the value in Sharedmemory2 to find the offset in SharedMemory1 to update the values. Finally, we make each child process exit, so that the child processes do not start looping.

Meanwhile, there is another loop, which allows the parent process to wait for all child processes to terminate. Once they all have terminated, the parent process opens its own Output log file, and retrieves the perfect numbers and the processes that identified it from SharedMemory1. It does so by using the offset in SharedMemory2 as a terminating condition for the loop, and then retrieves the pair of Perfect Number, and the process associated with it, and writes the same to the output file.

Analysis:

We see that creating multiple processes helps accelerate the process by dividing the tasks amongst multiple processes that run simultaneously. In fact, the speedup is almost ‘K’ times, where K is the number of processes. However, it is slightly less than ‘K’ due to some overhead involved in keeping all the processes running smoothly. As ‘N’ increases, a single process program begins to take lot of time as we run a for loop checking for each number.

Moreover, we know in most cases, fork() results in child processes who get their own share of the resources, and an independent copy of memory, such that any change imparted by the child processes does not carry forward to the parent processes. Thus, in this case, for the child processes to write to the memory about any perfect number they find, and for the parent process to be able to read it, we require a shared memory, which is achieved through shm_open().