COMP 8005 – Network and Security Applications Development

Assignment One

Multiple Processes Vs Multiple Threads

Kuanysh Boranbayev A00978728

1-20-2020

Contents

Objective

To design and implement two separate programs, one will use multiple processes and the other use multiple threads. The objective here is to obtain a measure of efficiency and performance of each implementation as they perform a set of specified tasks.

Overview

For the intensive mathematical task, computation of Narcissistic numbers within defined range is chosen. About Narcissistic numbers also known as Armstrong numbers, perfect digital invariant (Madachy 1979), or plus perfect number.

An ***n***-digit number that is the sum of the nth powers of its digits is called an ***n***-narcissistic number.

Examples: 153 = $1^3 + 5^3 + 3^3$,   370 = $3^3 + 7^3 + 0^3$

It can easily be shown that base-10 ***n***-narcissistic numbers can exist only for $\boldsymbol{n} \leq 60$, since

$$n \times 9^n < 10^{n-1}$$

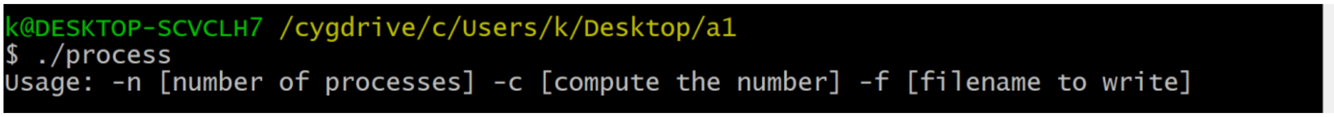
About Programs

Each program is designed so they execute and perform the same tasks but one with uses the number of child processes and the other threads.

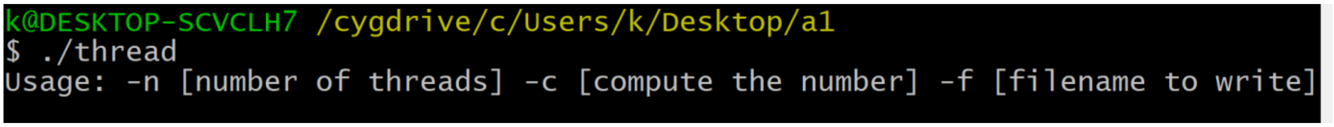
process.c – as name implies, the program creates child processes. Each child process will be given a task to compute and determine Narcissistic numbers within specified range.

thread.c – as name implies, the program creates n number of threads. Each thread will be given a task to compute and determine Narcissistic numbers within specified range.

Both programs accept the same number of arguments in the same order. Note: the programs are developed with GCC version 9.2.



```
k@DESKTOP-SCVCLH7 /cygdrive/c/Users/k/Desktop/a1
$ ./process
Usage: -n [number of processes] -c [compute the number] -f [filename to write]
```



```
k@DESKTOP-SCVCLH7 /cygdrive/c/Users/k/Desktop/a1
$ ./thread
Usage: -n [number of threads] -c [compute the number] -f [filename to write]
```

-n – flag for the number of processes or threads to create.
-c – a number that will be the end for the narcissistic computation. Both programs start computation from 1.
-f – a file name to write the results for further analysis such as import file to graph.

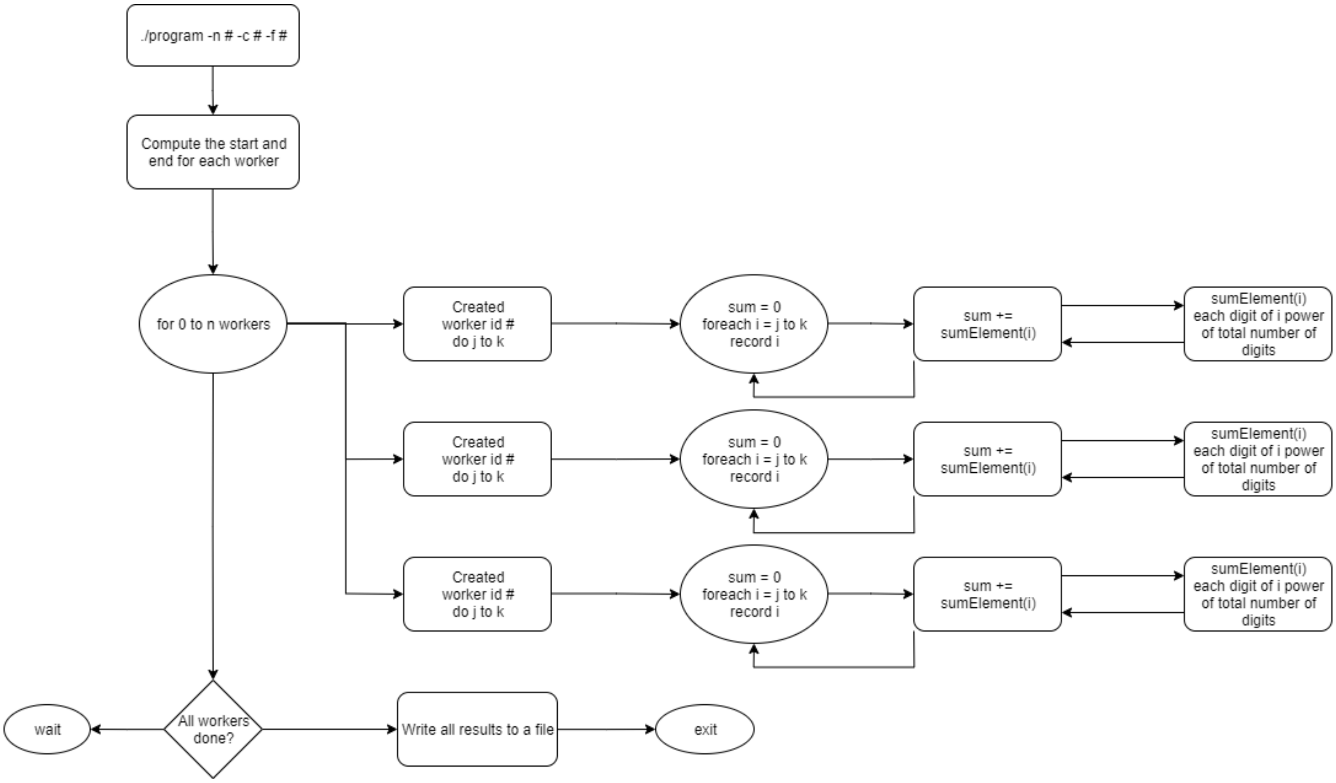
Once, the program is given all the necessary arguments, it will create specified number of “workers.” Program will determine so that each worker will be given equal range to compute with different intervals such as 1st worker will compute from 1 to j, 2nd worker will compute from j – k, 3rd worker will compute k – l and so on.

Parent process will create a shared memory in virtual environment. The shared memory will store array of data. Each data will keep the id of the worker, the range given to that worker, a time, and a set of found narcissistic numbers.

In addition, parent process will also keep its time of execution from start to end.

Diagram

Note: the program does not limit itself to create only 3 workers as shown below.

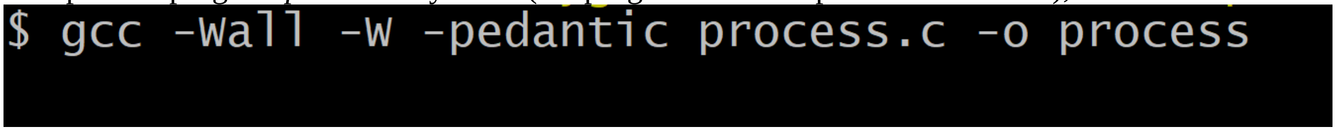

./program -n # -c # -f #
Compute the start and end for each worker
for 0 to n workers
Created worker id # do j to k
sum = 0 foreach i = j to k record i
sum += sumElement(i)
sumElement(i) each digit of i power of total number of digits
Created worker id # do j to k
sum = 0 foreach i = j to k record i
sum += sumElement(i)
sumElement(i) each digit of i power of total number of digits
Created worker id # do j to k
sum = 0 foreach i = j to k record i
sum += sumElement(i)
sumElement(i) each digit of i power of total number of digits
wait
All workers done?
Write all results to a file
exit

Usage and Experiment

In this section, the usage of the both programs will be shown. And the results will be analyzed.

Process.c

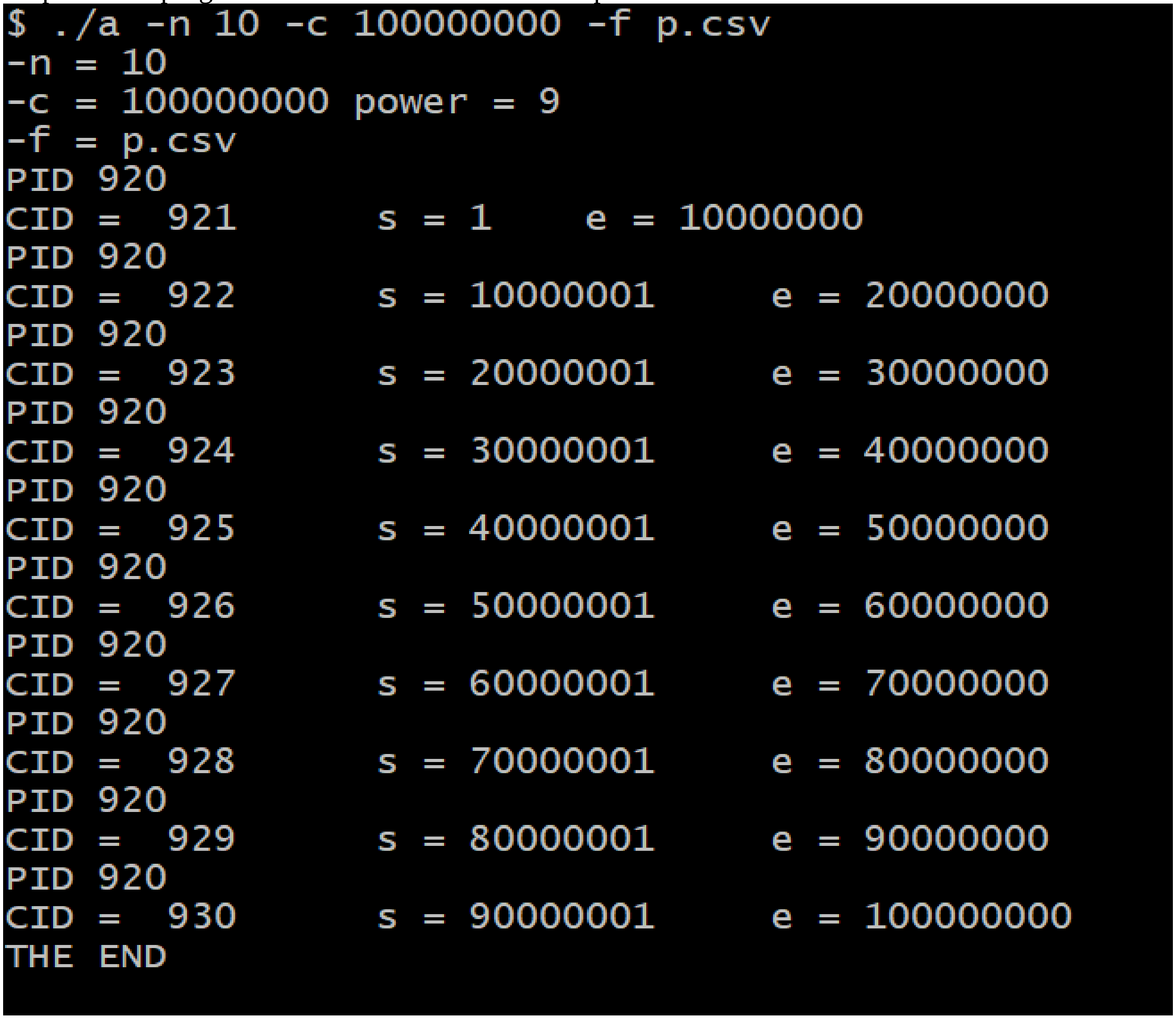
Compile the program *process.c* by GCC (the program is developed on version 9.2), as shown below



```
$ gcc -Wall -W -pedantic process.c -o process
```

Then, execute the program as shown below:

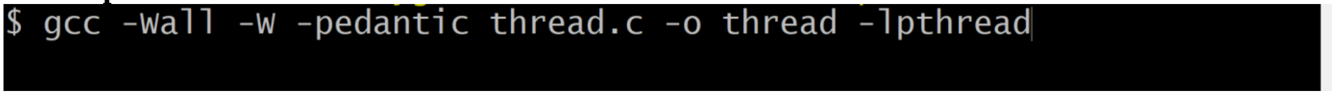
There are some minor differences: “. /a” is the compiled program executable with different name;
“-n 10” set program to create 10 child processes;
“-c 100000000” set program to find narcissistic number between 1 and 10^8.
“-f p.csv” set program to create a file with a name “p.csv” and write the results into it.



```
$ ./a -n 10 -c 100000000 -f p.csv
-n = 10
-c = 100000000 power = 9
-f = p.csv
PID 920
CID =  921      s = 1    e = 10000000
PID 920
CID =  922      s = 10000001     e = 20000000
PID 920
CID =  923      s = 20000001     e = 30000000
PID 920
CID =  924      s = 30000001     e = 40000000
PID 920
CID =  925      s = 40000001     e = 50000000
PID 920
CID =  926      s = 50000001     e = 60000000
PID 920
CID =  927      s = 60000001     e = 70000000
PID 920
CID =  928      s = 70000001     e = 80000000
PID 920
CID =  929      s = 80000001     e = 90000000
PID 920
CID =  930      s = 90000001     e = 100000000
THE END
```

thread.c

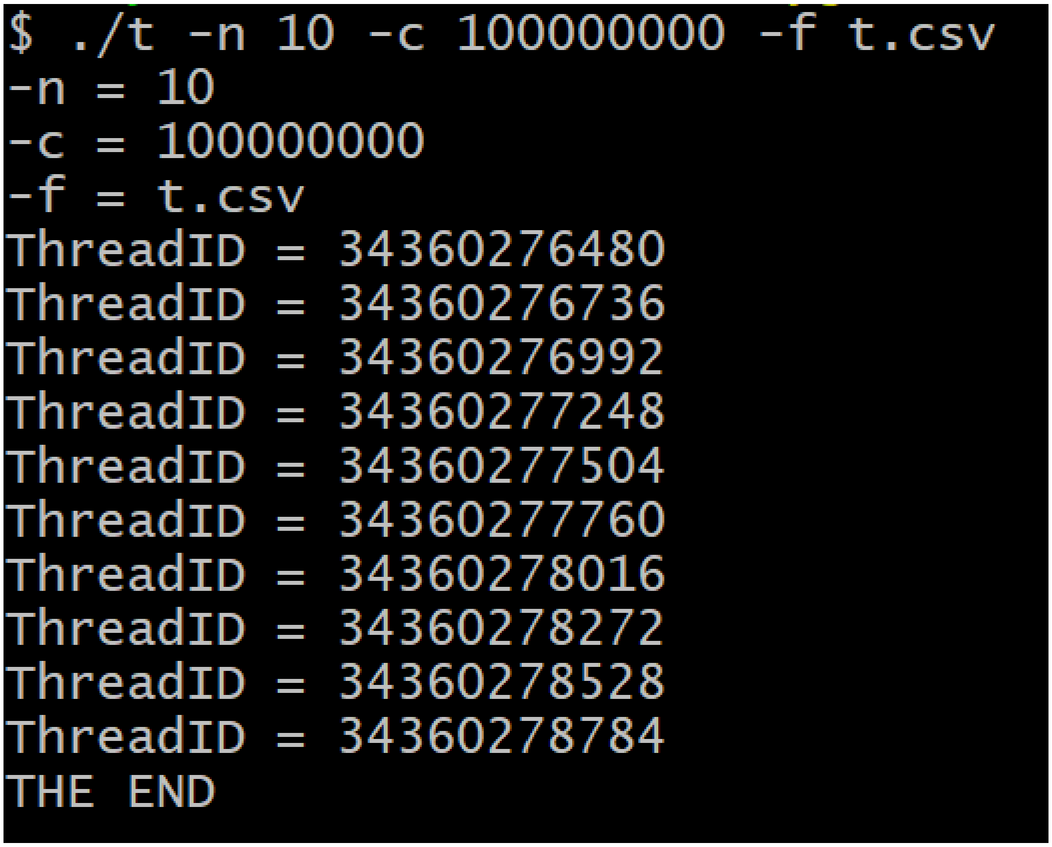
Compile the program *thread.c* by GCC (the program is developed on version 9.2), as shown below.
Note: **-lpthread**



```
$ gcc -Wall -W -pedantic thread.c -o thread -lpthread
```

Then, execute the program as shown below:

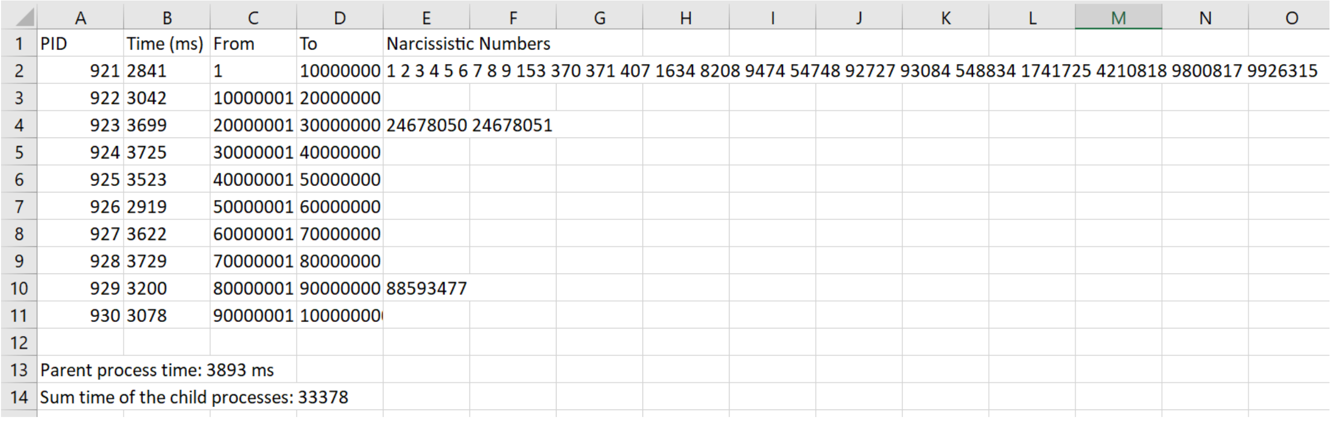
There are some minor differences: “. /t” is the compiled program executable with different name;
“-n 10” set program to create 10 threads;
“-c 100000000” set program to find narcissistic number between 1 and 10^8.
“-f t.csv” set program to create a file with a name “t.csv” and write the results into it.



```
$ ./t -n 10 -c 100000000 -f t.csv
-n = 10
-c = 100000000
-f = t.csv
ThreadID = 34360276480
ThreadID = 34360276736
ThreadID = 34360276992
ThreadID = 34360277248
ThreadID = 34360277504
ThreadID = 34360277760
ThreadID = 34360278016
ThreadID = 34360278272
ThreadID = 34360278528
ThreadID = 34360278784
THE END
```

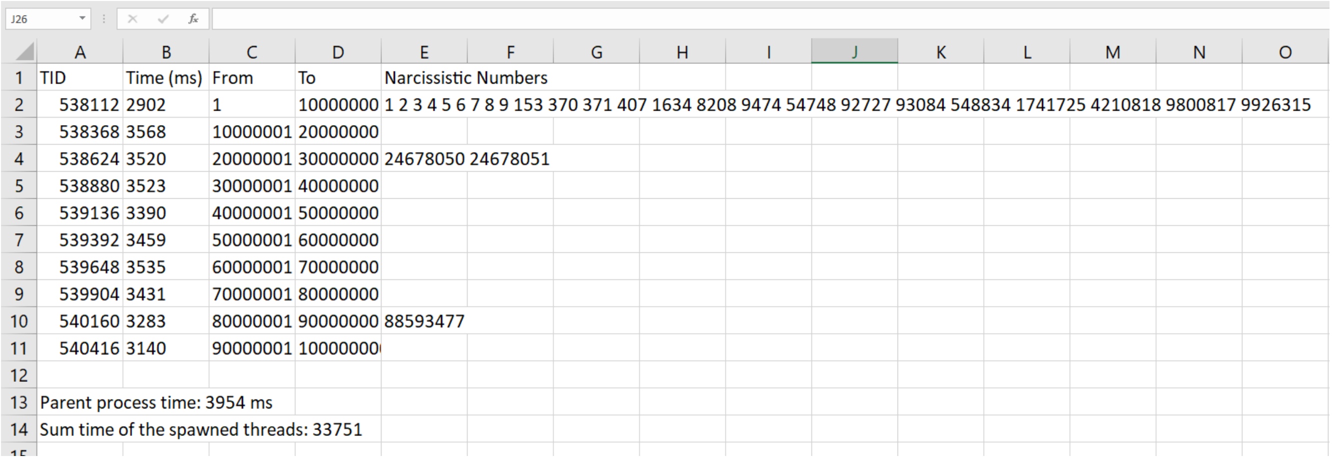
Open each written file and import it to any office tool such as MS Office Excel or LibreOffice Calc.

Contents of the “p.csv”



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	PID	Time (ms)	From	To	Narcissistic Numbers										
2	921	2841	1	10000000	1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474 54748 92727 93084 548834 1741725 4210818 9800817 9926315										
3	922	3042	10000001	20000000											
4	923	3699	20000001	30000000	24678050 24678051										
5	924	3725	30000001	40000000											
6	925	3523	40000001	50000000											
7	926	2919	50000001	60000000											
8	927	3622	60000001	70000000											
9	928	3729	70000001	80000000											
10	929	3200	80000001	90000000	88593477										
11	930	3078	90000001	10000000											
12															
13	Parent process time: 3893 ms														
14	Sum time of the child processes: 33378														

Contents of the “t.csv”



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	TID	Time (ms)	From	To	Narcissistic Numbers										
2	538112	2902	1	10000000	1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474 54748 92727 93084 548834 1741725 4210818 9800817 9926315										
3	538368	3568	10000001	20000000											
4	538624	3520	20000001	30000000	24678050 24678051										
5	538880	3523	30000001	40000000											
6	539136	3390	40000001	50000000											
7	539392	3459	50000001	60000000											
8	539648	3535	60000001	70000000											
9	539904	3431	70000001	80000000											
10	540160	3283	80000001	90000000	88593477										
11	540416	3140	90000001	10000000											
12															
13	Parent process time: 3954 ms														
14	Sum time of the spawned threads: 33751														

As seen above, both programs executed and provided similar results for the given experiment.

From the data, we see PID and TID which stand for process id and thread id, respectively.

Time column indicates the time for completion of the worker. It’s given in ms, milliseconds, 10^{-3} seconds.

From and To indicate the range the worker was assigned to compute.

Under Narcissistic Numbers column, we see found numbers that relate to that characteristics.

Parent process time indicate the main program completion time. Note, it’s quite different from the sum of the child processes or threads time. As the program creates a worker, each worker will immediately starts computing at different pace from other.

So, from the results above, we see that process results show times fluctuating between 2.8 s and 3.73 s whereas thread results show times fluctuating between 2.9 s and 3.6 s.

Also note in process results, the first process id 921 and the 6th process id 926. They both show similar time ~2.9 s meaning CPU executed them first than others. Processes with larger delays indicate they exited last from CPU cores.

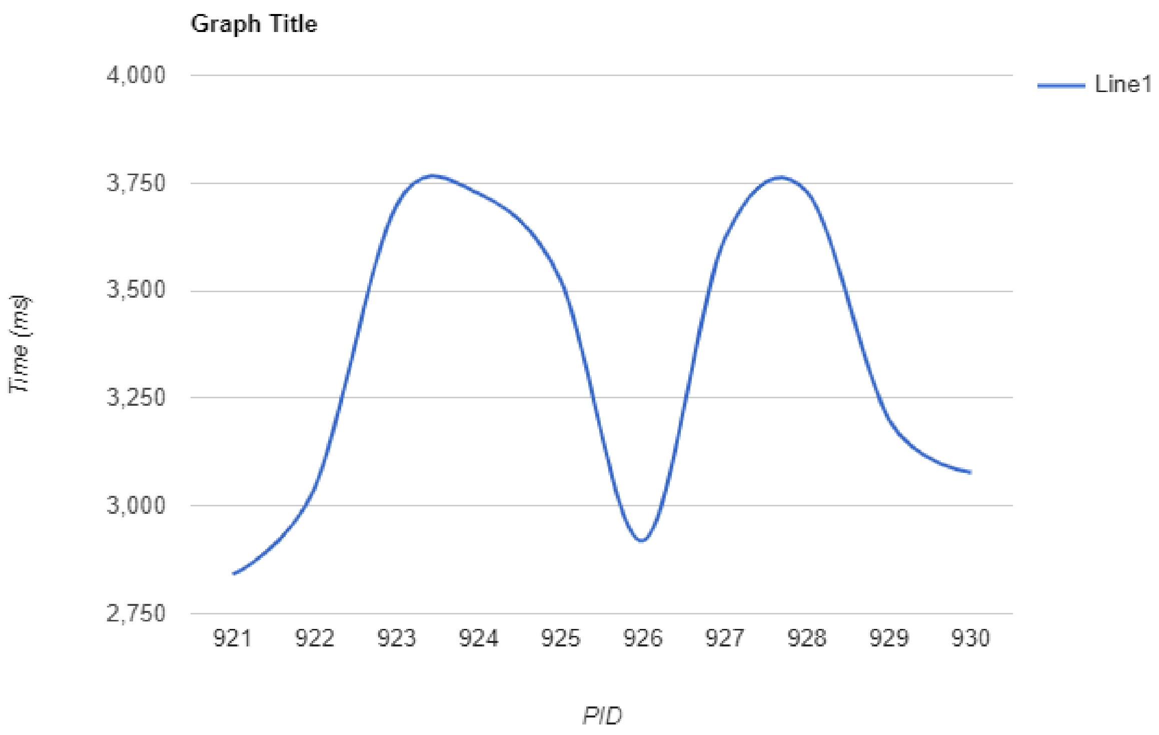
As for thread results, we see similar times among different threads except first one. It can be explained as the first one is faster because each process in CPU is allowed to use only thread. As for the others, it can be explained as the program is asking the CPU for additional slots to execute.

In overall results, we see that process is winning only by a little bit margin. Although, that is the case, threads are executing at the similar pace without occupying the whole core for computation.

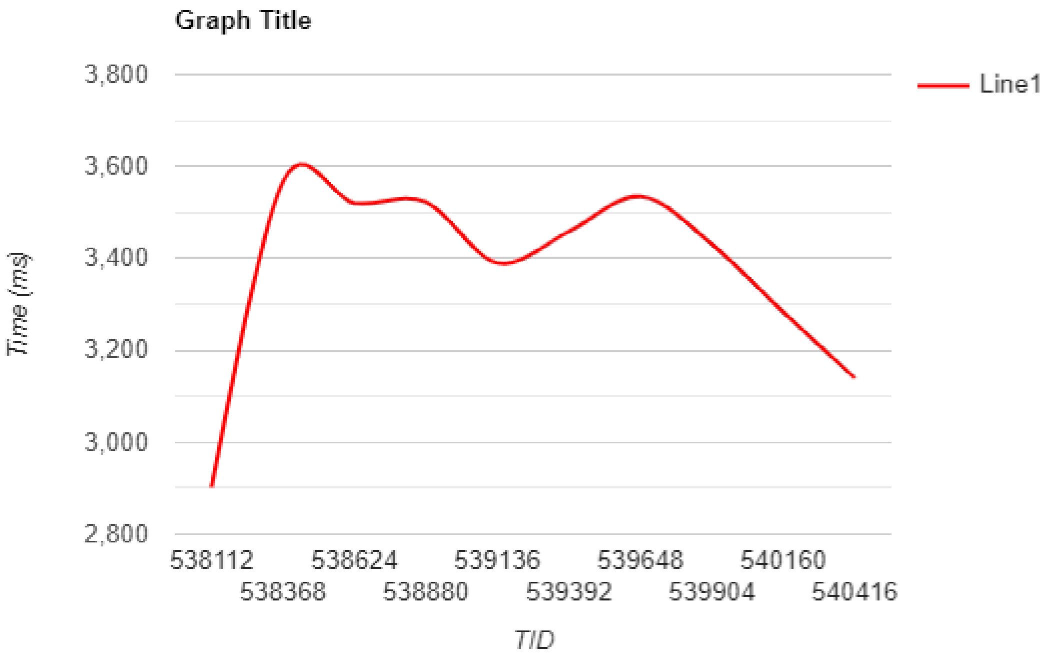
The situation might change however if the CPU with QuadCore will not be able to create additional virtual CPUs like the one used in this experiment. In that case CPU will be able to use only maximum of 4 processes at a time, as each process require the whole CPU slot unlike lightweight threads that do not and execute with less power consumption.

Graphs

Process Graph



Thread Graph



References

http://mathworld.wolfram.com/NarcissisticNumber.html