CS307

Operating Systems

Project Report

Concurrent Programming - Processes and Threads

Professor:

Khaldoun Al Khalidi

Students:

Imamović Kenan 200302016

Kadić Amina 200302010

Sarajevo, December 2022

Contents

Table of figures 2
1. Abstract 3
2. Project Plan 3
2.1 Requirements 3
2.2 Program Logic 3
3. Implementation 4
3.1 Methods used 4
3.2 Method calculating primes 4
3.3 Main function 5
4. Testing 6
4.1 Test plan 6
4.2 Single slave testing 6
4.3 Parallel testing 7
5. Conclusion 9

Table of figures

Table 1: Methods used 4
Table 2: single slave testing 6
Table 3: comparison between threads and processes with the same configurations 7

Figure 1: single thread testing chart (max number vs time) 6
Figure 2: single process testing chart (max number vs time) 6
Figure 3: processes vs time chart 7
Figure 4: 32 threads configuration 8
Figure 5: 32 processes configuration 8
Figure 6: 64 processes configuration pt.1 8
Figure 7: 64 threads configuration pt.1 8
Figure 8: 64 threads configuration pt.2 8
Figure 9: 64 threads configuration pt.2 8

1. Abstract

This project aims to compare the performance of using threads and processes to calculate the number of prime numbers within a specified range. The program allows the user to input the number of threads or processes to use, the maximum number to check for primality, and the choice between using threads or processes. The program then creates the specified number of threads or processes and divides the range of numbers evenly among them. Each thread or process calculates the number of prime numbers within its assigned range and returns the result. The main thread or process then sums the results and prints out the total number of prime numbers found and the time it took each slave to finish its execution. The program also measures the elapsed time and prints out the results.

2. Project Plan

2.1 Requirements

In this project we will use the Master/Slave model to compute prime numbers using two methods:

- Forking children;
- Threads

This program is a simple example of how to use threads or processes to parallelize a task. It counts the number of prime numbers in a given range, using either threads or processes, depending on the user's choice. The project will be written in C language, and tested using the Cygwin terminal.

2.2 Program Logic

Each master program, either via thread or fork, creates a certain number of slaves, which we will call N. Each slave process has its slave_id (0,1,2,...N-1) which will be used for computation of prime numbers. The prime numbers will be calculated in a certain interval which has a maximum called max_prime. Each number is computed starting from 3+2*slave_id (initial slave_id) up to the maximum in increments of 2*num_slaves which is the total number of slave processes/threads created.

The program requires a function that will calculate the primes, and the main function that will create threads and proceses which will call the function.

The calculate_primes function is defined next .It takes a void pointer as an argument, and because of that we will define a struct called DataSection including an integer „id“ containing slave_id's for each thread/process, integer „num_threads“ containing the number of threads/processes the user wants to create , and an unsigned long „max_prime“ which will be a stopping point for our searching interval.

The function first casts this argument to a pointer to the DataSection structure, and then extracts the data from the structure. It initializes some variables and sets the starting number based on the thread or process ID. The function then enters a loop that iterates through the range of numbers to check for primality. For each number, it checks for divisibility by all integers from 2 to the square root of the number, and if the number is not divisible and is greater than 1, it increments the primes counter. The function then increments the number by twice the number of threads or processes, to skip over numbers that have already been checked. When the loop finishes, the function returns the number of prime numbers found as a void pointer.

The main function is then defined, which first records the current time using the clock function and initializes some variables. It then prompts the user to enter the number of threads or processes to use, the maximum number to check for primality, and whether to use threads or processes. The program then enters an if statement that checks the value of choice. If the user has chosen to use processes, the program creates a number of child processes using the fork function and assigns their IDs to an array.

Each child process then calls the calculate_primes function, passing a DataSection structure containing its ID, the total number of processes, and the maximum number to check for primality. After the calculate_primes function returns, the child process prints the number of prime numbers it found and exits. The parent process waits for all child processes to finish using the wait function.

If the user has chosen to use threads, the program creates a number of threads using the pthread_create function, passing the address of each thread, a NULL value, the calculate_primes function, and a DataSection structure containing the thread's ID, the total number of threads, and the maximum number to check for primality as arguments. After all threads are created, the program enters a loop that iterates through the threads and waits for each one to finish using the pthread_join function. When all threads have finished, the program calculates the elapsed time using the clock function and prints the total number of prime numbers found. Finally, the main function returns 0 to indicate successful execution.

3. Implementation

3.1 Methods used

Table 1: Methods used

Method	Use	Library
pthread_create()	Create a thread	pthread.h
pthread_join()	Join a thread to finish execution	pthread.h
clock()	Measure clock ticks	time.h
fork()	Create a child process	unistd.h
wait()	Wait until a process is finished	wait.h
scanf()	Scan input	stdio.h
printf()	Print output	stdio.h
strcmp()	Compare strings	string.h

3.2 Method calculating primes

```
void *calculate_primes(void *arg) {
    struct arg *data = (struct arg *)arg;
    int slave_id = data->id;
    int num_slaves = data->numberOfSlaves;
    unsigned long MaxPrime = data->MaxPrime;
    unsigned long num;
    int i, flag;
    num = 3 + 2 * slave_id;
    unsigned long count = 0;
    while (num <= MaxPrime) {
        flag = 0;
        for (i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                flag = 1;
                break;
            }
        }
        if (flag == 0 && (num > 1)) {
            ++count;
        }
        num += 2 * num_slaves;
    }
    return (void *)count;
}
```

3.3 Main function

```
int main() {
    clock_t start, end;
    double total;
    static double starter;
    static double ender;
    static struct tms timTwo;
    static struct tms timOne;
    start = clock();
    printf("Enter the number of slaves: ");
    scanf("%d", &Number);
    printf("Enter the maximum prime to find: ");
    scanf("%lu", &MaxThr);
    printf("Enter 'thread' or 'process': ");
    scanf("%s", &choice);
    if ((strcmp(choice, "process")) == 0) {
        pid_t ids[Number];
        unsigned long count;
        for (int i = 0; i < Number; i++) {
            starter = times(&timOne);
            ids[i] = fork();
            if (ids[i] == 0) {
                struct arg data;
                data.id = i;
                data.numberOfSlaves = Number;
                data.MaxPrime = MaxThr;
                count = (unsigned
long)calculate_primes(&data);
                ender = times(&timTwo);
                total = ((double)(ender - starter) /
CLOCKS_PER_SEC)*1000;
                printf("Process %d found %lu prime
numbers in %f milliseconds.\n", i, count, total);
                return 0;
            } else if (ids[i] < 0) {
                printf("Fork was unsuccessful");
            }
        }
        for (int i = 0; i < Number; i++) {
            wait(NULL);
        }
    }
    else if ((strcmp(choice, "thread")) == 0) {
        pthread_t threads[Number];
        struct arg myData[Number];
        for (int i = 0; i < Number; i++) {
            myData[i].id = i;
            myData[i].numberOfSlaves = Number;
            myData[i].MaxPrime = MaxThr;
            starter = times(&timOne);
            pthread_create(&threads[i], NULL, (void
*)calculate_primes, &myData[i]);
        }
        for (int i = 0; i < Number; i++) {
            void *result;
            pthread_join(threads[i], &result);
            unsigned long count = (unsigned
long)result;
             ender = times(&timTwo);
             total = ((double)(ender - starter) /
CLOCKS_PER_SEC)*1000;
            printf("thread %d found %lu prime numbers
in %f milliseconds.\n", i, count, total); }
     }else {
        printf("Error: not found\n");
    }
    end = clock();
    total = ((double) (end - start) /
CLOCKS_PER_SEC) * 1000;
    printf("\nTotal elapsed time was %f
miliseconds\n",total  );
    return 0;
}
```

4. Testing

4.1 Test plan

The program will be tested in the following configurations

I: single slave

Number of children: 1

Maximum: 100, 1000, 10000, 100000, 1000000

II: parallel threads and processes

Number of slaves: 2, 4, 8, 16, 32, 64

Maximum: 1000000

4.2 Single slave testing

Table 2: single slave testing

	Primes found		Primes found		Primes found		Primes found		Primes found	
	Max=100	Time(ms)	Max=1000	Time(ms)	Max=10000	Time(ms)	Max=100000	Time(ms)	Max=1000000	Time(ms)
Thread	24	0	167	1	1228	13	9591	680	78497	65090
Process	24	11	167	12	1228	22	9591	691	78497	59605

If we wish to see this table as a chart in which we will compare the maximum number searched vs the time it took the program to search through the numbers it will look like this:

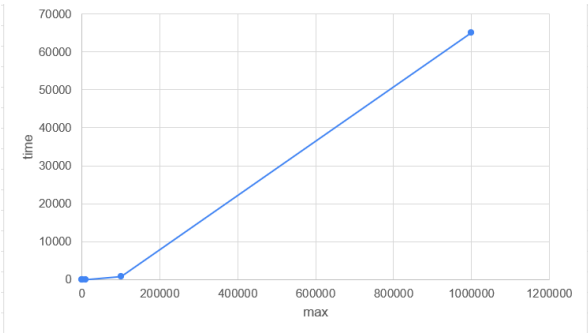

Figure 1: single thread testing chart (max number vs time)

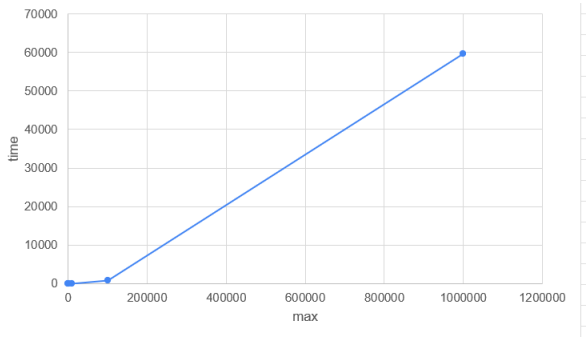

Figure 2: single process testing chart (max number vs time)

4.3 Parallel testing

For the sake of simplicity, we will only include configurations with 2,4,8 and 16 slaves into the table, and we will provide photos of the configurations with 32 and 64 slaves.

Table 3: comparison between threads and processes with the same configurations

	Number of slaves=2				Number of slaves=4				Number of slaves=8				Number of slaves =16			
slave	Primes found by thread	Time (ms)	Primes found by process	Time (ms)	Primes found by thread	Time (ms)	Primes found by process	Time (ms)	Primes found by thread	Time (ms)	Primes found by process	Time (ms)	Primes found by thread	Time (ms)	Primes found by process	Time (ms)
0	39322	41161	39322	34913	19653	20091	19653	20460	9838	12952	9838	12562	4928	12164	4928	6353
1	39175	411610	39175	35006	19623	20361	19623	20558	9816	13026	9816	12406	4889	12166	4889	6150
2					19669	20652	19669	20329	9832	13026	9832	12609	4899	12166	4899	6261
3					19552	20652	19552	20260	9791	13026	9791	12519	4884	12166	4884	10309
4									9815	13026	9815	12554	4894	12166	4894	8963
5									9807	13026	9807	12587	4891	12166	4891	12594
6									9837	13026	9837	12595	4927	12195	4927	12643
7									9761	13026	9761	12500	4849	12195	4849	12571
8													4910	12195	4910	10099
9													4927	12328	4927	12358
10													4933	12333	4933	12706
11													4907	12333	4907	12669
12													4921	12333	4921	12202
13													4916	12536	4916	8928
14													4910	12595	4910	11746
15													4912	12595	4912	9427

If we represent the process number vs total time it will look like this:

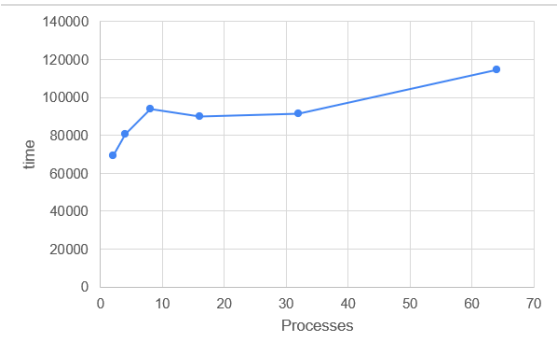

Figure 3: processes vs time chart

Because of thread time inconsistencies, we will not include a thread comparison

```
Enter the number of slaves: 32
Enter the maximum prime to find: 1000000
Enter 'thread' or 'process': thread
thread 0 found 2441 prime numbers in 5635.000000 milliseconds.
thread 1 found 2460 prime numbers in 5635.000000 milliseconds.
thread 2 found 2441 prime numbers in 5635.000000 milliseconds.
thread 3 found 2433 prime numbers in 5635.000000 milliseconds.
thread 4 found 2451 prime numbers in 5635.000000 milliseconds.
thread 5 found 2433 prime numbers in 5635.000000 milliseconds.
thread 6 found 2439 prime numbers in 5635.000000 milliseconds.
thread 7 found 2407 prime numbers in 5635.000000 milliseconds.
thread 8 found 2449 prime numbers in 6025.000000 milliseconds.
thread 9 found 2458 prime numbers in 6025.000000 milliseconds.
thread 10 found 2470 prime numbers in 6145.000000 milliseconds.
thread 11 found 2457 prime numbers in 6301.000000 milliseconds.
thread 12 found 2491 prime numbers in 6614.000000 milliseconds.
thread 13 found 2459 prime numbers in 6715.000000 milliseconds.
thread 14 found 2424 prime numbers in 6715.000000 milliseconds.
thread 15 found 2456 prime numbers in 6850.000000 milliseconds.
thread 16 found 2487 prime numbers in 7280.000000 milliseconds.
thread 17 found 2429 prime numbers in 7280.000000 milliseconds.
thread 18 found 2458 prime numbers in 7396.000000 milliseconds.
thread 19 found 2451 prime numbers in 7454.000000 milliseconds.
thread 20 found 2443 prime numbers in 7672.000000 milliseconds.
thread 21 found 2458 prime numbers in 7776.000000 milliseconds.
thread 22 found 2488 prime numbers in 7845.000000 milliseconds.
thread 23 found 2442 prime numbers in 7923.000000 milliseconds.
thread 24 found 2461 prime numbers in 8040.000000 milliseconds.
thread 25 found 2469 prime numbers in 8100.000000 milliseconds.
thread 26 found 2463 prime numbers in 8149.000000 milliseconds.
thread 27 found 2450 prime numbers in 8186.000000 milliseconds.
thread 28 found 2430 prime numbers in 8268.000000 milliseconds.
thread 29 found 2457 prime numbers in 8365.000000 milliseconds.
thread 30 found 2486 prime numbers in 8619.000000 milliseconds.
thread 31 found 2456 prime numbers in 8629.000000 milliseconds.

Total elapsed time was 94078.000000 miliseconds
```

Figure 4: 32 threads configuration

```
Enter the number of slaves: 32
Enter the maximum prime to find: 1000000
Enter 'thread' or 'process': process
Process 2 found 2441 prime numbers in 3092.000000 milliseconds.
Process 0 found 2441 prime numbers in 3110.000000 milliseconds.
Process 4 found 2451 prime numbers in 3080.000000 milliseconds.
Process 1 found 2460 prime numbers in 3171.000000 milliseconds.
Process 11 found 2457 prime numbers in 5403.000000 milliseconds.
Process 14 found 2424 prime numbers in 5320.000000 milliseconds.
Process 18 found 2458 prime numbers in 5140.000000 milliseconds.
Process 24 found 2461 prime numbers in 5877.000000 milliseconds.
Process 3 found 2433 prime numbers in 7441.000000 milliseconds.
Process 7 found 2407 prime numbers in 7561.000000 milliseconds.
Process 10 found 2470 prime numbers in 7716.000000 milliseconds.
Process 27 found 2450 prime numbers in 4754.000000 milliseconds.
Process 15 found 2456 prime numbers in 9128.000000 milliseconds.
Process 20 found 2443 prime numbers in 8810.000000 milliseconds.
Process 30 found 2486 prime numbers in 5829.000000 milliseconds.
Process 22 found 2488 prime numbers in 9140.000000 milliseconds.
Process 8 found 2449 prime numbers in 10543.000000 milliseconds.
Process 9 found 2458 prime numbers in 10610.000000 milliseconds.
Process 12 found 2491 prime numbers in 11006.000000 milliseconds.
Process 5 found 2433 prime numbers in 11341.000000 milliseconds.
Process 16 found 2487 prime numbers in 11147.000000 milliseconds.
Process 17 found 2429 prime numbers in 11362.000000 milliseconds.
Process 6 found 2439 prime numbers in 11913.000000 milliseconds.
Process 13 found 2459 prime numbers in 11901.000000 milliseconds.
Process 21 found 2458 prime numbers in 11320.000000 milliseconds.
Process 19 found 2451 prime numbers in 11626.000000 milliseconds.
Process 25 found 2469 prime numbers in 10035.000000 milliseconds.
Process 23 found 2442 prime numbers in 11154.000000 milliseconds.
Process 28 found 2430 prime numbers in 8869.000000 milliseconds.
Process 31 found 2456 prime numbers in 8347.000000 milliseconds.
Process 29 found 2457 prime numbers in 8979.000000 milliseconds.
Process 26 found 2463 prime numbers in 9600.000000 milliseconds.

Total elapsed time was 91571.000000 miliseconds
```

Figure 5: 32 processes configuration

```
Process 0 found 1226 prime numbers in 1954.000000 milliseconds.
Process 1 found 1220 prime numbers in 1946.000000 milliseconds.
Process 2 found 1223 prime numbers in 1980.000000 milliseconds.
Process 3 found 1245 prime numbers in 2007.000000 milliseconds.
Process 17 found 1203 prime numbers in 3082.000000 milliseconds.
Process 19 found 1218 prime numbers in 2865.000000 milliseconds.
Process 18 found 1218 prime numbers in 3073.000000 milliseconds.
Process 20 found 1219 prime numbers in 3469.000000 milliseconds.
Process 10 found 1225 prime numbers in 4732.000000 milliseconds.
Process 4 found 1223 prime numbers in 5291.000000 milliseconds.
Process 30 found 1258 prime numbers in 3124.000000 milliseconds.
Process 33 found 1240 prime numbers in 2344.000000 milliseconds.
Process 28 found 1220 prime numbers in 4201.000000 milliseconds.
Process 16 found 1254 prime numbers in 6013.000000 milliseconds.
Process 11 found 1222 prime numbers in 6319.000000 milliseconds.
Process 35 found 1188 prime numbers in 3678.000000 milliseconds.
Process 32 found 1215 prime numbers in 4755.000000 milliseconds.
Process 6 found 1209 prime numbers in 7848.000000 milliseconds.
Process 38 found 1230 prime numbers in 3393.000000 milliseconds.
Process 40 found 1215 prime numbers in 3142.000000 milliseconds.
Process 21 found 1240 prime numbers in 7284.000000 milliseconds.
Process 29 found 1233 prime numbers in 7074.000000 milliseconds.
Process 36 found 1228 prime numbers in 5512.000000 milliseconds.
Process 41 found 1236 prime numbers in 4171.000000 milliseconds.
Process 43 found 1235 prime numbers in 3830.000000 milliseconds.
Process 8 found 1234 prime numbers in 9655.000000 milliseconds.
Process 22 found 1258 prime numbers in 9098.000000 milliseconds.
Process 13 found 1238 prime numbers in 10109.000000 milliseconds.
Process 37 found 1203 prime numbers in 6541.000000 milliseconds.
Process 47 found 1223 prime numbers in 4474.000000 milliseconds.
Process 26 found 1254 prime numbers in 9347.000000 milliseconds.
Process 12 found 1244 prime numbers in 11088.000000 milliseconds.
Process 25 found 1238 prime numbers in 9872.000000 milliseconds.
Process 7 found 1199 prime numbers in 11405.000000 milliseconds.
Process 5 found 1230 prime numbers in 11718.000000 milliseconds.
Process 9 found 1222 prime numbers in 11702.000000 milliseconds.
Process 24 found 1236 prime numbers in 10498.000000 milliseconds.
Process 34 found 1218 prime numbers in 8613.000000 milliseconds.
Process 52 found 1224 prime numbers in 4069.000000 milliseconds.
Process 14 found 1209 prime numbers in 12097.000000 milliseconds.
Process 23 found 1202 prime numbers in 11597.000000 milliseconds.
Process 61 found 1224 prime numbers in 2363.000000 milliseconds.
Process 15 found 1233 prime numbers in 12923.000000 milliseconds.
Process 48 found 1233 prime numbers in 6208.000000 milliseconds.
Process 63 found 1223 prime numbers in 2759.000000 milliseconds.
Process 56 found 1225 prime numbers in 4513.000000 milliseconds.
Process 46 found 1215 prime numbers in 7654.000000 milliseconds.
Process 27 found 1235 prime numbers in 12195.000000 milliseconds.
Process 44 found 1247 prime numbers in 8398.000000 milliseconds.
Process 31 found 1233 prime numbers in 12029.000000 milliseconds.
Process 57 found 1231 prime numbers in 5352.000000 milliseconds.
Process 39 found 1208 prime numbers in 10381.000000 milliseconds.
Process 45 found 1221 prime numbers in 9106.000000 milliseconds.
Process 60 found 1210 prime numbers in 4970.000000 milliseconds.
Process 59 found 1215 prime numbers in 5297.000000 milliseconds.
Process 58 found 1209 prime numbers in 5564.000000 milliseconds.
Process 42 found 1245 prime numbers in 10066.000000 milliseconds.
Process 50 found 1240 prime numbers in 7998.000000 milliseconds.
Process 51 found 1233 prime numbers in 7856.000000 milliseconds.
Process 53 found 1218 prime numbers in 7443.000000 milliseconds.
```

Figure 6: 64 processes configuration pt.1

```
thread 0 found 1226 prime numbers in 135.000000 milliseconds.
thread 1 found 1220 prime numbers in 135.000000 milliseconds.
thread 2 found 1223 prime numbers in 135.000000 milliseconds.
thread 3 found 1245 prime numbers in 135.000000 milliseconds.
thread 4 found 1223 prime numbers in 135.000000 milliseconds.
thread 5 found 1230 prime numbers in 135.000000 milliseconds.
thread 6 found 1209 prime numbers in 135.000000 milliseconds.
thread 7 found 1199 prime numbers in 135.000000 milliseconds.
thread 8 found 1234 prime numbers in 135.000000 milliseconds.
thread 9 found 1222 prime numbers in 135.000000 milliseconds.
thread 10 found 1225 prime numbers in 135.000000 milliseconds.
thread 11 found 1222 prime numbers in 135.000000 milliseconds.
thread 12 found 1244 prime numbers in 135.000000 milliseconds.
thread 13 found 1238 prime numbers in 135.000000 milliseconds.
thread 14 found 1209 prime numbers in 135.000000 milliseconds.
thread 15 found 1233 prime numbers in 135.000000 milliseconds.
thread 16 found 1254 prime numbers in 135.000000 milliseconds.
thread 17 found 1203 prime numbers in 135.000000 milliseconds.
thread 18 found 1218 prime numbers in 135.000000 milliseconds.
thread 19 found 1218 prime numbers in 135.000000 milliseconds.
thread 20 found 1219 prime numbers in 135.000000 milliseconds.
thread 21 found 1240 prime numbers in 135.000000 milliseconds.
thread 22 found 1258 prime numbers in 135.000000 milliseconds.
thread 23 found 1202 prime numbers in 135.000000 milliseconds.
thread 24 found 1236 prime numbers in 135.000000 milliseconds.
thread 25 found 1238 prime numbers in 135.000000 milliseconds.
thread 26 found 1254 prime numbers in 135.000000 milliseconds.
thread 27 found 1235 prime numbers in 135.000000 milliseconds.
thread 28 found 1220 prime numbers in 135.000000 milliseconds.
thread 29 found 1233 prime numbers in 135.000000 milliseconds.
thread 30 found 1258 prime numbers in 135.000000 milliseconds.
thread 31 found 1233 prime numbers in 135.000000 milliseconds.
thread 32 found 1215 prime numbers in 135.000000 milliseconds.
thread 33 found 1240 prime numbers in 135.000000 milliseconds.
thread 34 found 1218 prime numbers in 135.000000 milliseconds.
thread 35 found 1188 prime numbers in 135.000000 milliseconds.
thread 36 found 1228 prime numbers in 135.000000 milliseconds.
thread 37 found 1203 prime numbers in 135.000000 milliseconds.
thread 38 found 1230 prime numbers in 135.000000 milliseconds.
thread 39 found 1208 prime numbers in 135.000000 milliseconds.
thread 40 found 1215 prime numbers in 365.000000 milliseconds.
thread 41 found 1236 prime numbers in 603.000000 milliseconds.
thread 42 found 1245 prime numbers in 992.000000 milliseconds.
thread 43 found 1235 prime numbers in 992.000000 milliseconds.
thread 44 found 1247 prime numbers in 1195.000000 milliseconds.
thread 45 found 1221 prime numbers in 1409.000000 milliseconds.
thread 46 found 1215 prime numbers in 1597.000000 milliseconds.
thread 47 found 1223 prime numbers in 1597.000000 milliseconds.
thread 48 found 1233 prime numbers in 1597.000000 milliseconds.
thread 49 found 1226 prime numbers in 1941.000000 milliseconds.
thread 50 found 1240 prime numbers in 2134.000000 milliseconds.
thread 51 found 1233 prime numbers in 2134.000000 milliseconds.
thread 52 found 1224 prime numbers in 2134.000000 milliseconds.
thread 53 found 1218 prime numbers in 2344.000000 milliseconds.
thread 54 found 1230 prime numbers in 2392.000000 milliseconds.
thread 55 found 1240 prime numbers in 2413.000000 milliseconds.
thread 56 found 1225 prime numbers in 2512.000000 milliseconds.
thread 57 found 1231 prime numbers in 2602.000000 milliseconds.
thread 58 found 1209 prime numbers in 2668.000000 milliseconds.
thread 59 found 1215 prime numbers in 2668.000000 milliseconds.
thread 60 found 1210 prime numbers in 2685.000000 milliseconds.
```

Figure 7: 64 threads configuration pt.1

```
Process 49 found 1226 prime numbers in 8391.000000 milliseconds.
Process 54 found 1230 prime numbers in 7235.000000 milliseconds.
Process 55 found 1240 prime numbers in 7075.000000 milliseconds.
Process 62 found 1228 prime numbers in 5189.000000 milliseconds.

Total elapsed time was 114372.000000 miliseconds
```

Figure 8: 64 threads configuration pt.2

```
thread 61 found 1224 prime numbers in 2759.000000 milliseconds.
thread 62 found 1228 prime numbers in 2850.000000 milliseconds.
thread 63 found 1223 prime numbers in 2850.000000 milliseconds.

Total elapsed time was 95609.000000 miliseconds
```

Figure 9: 64 threads configuration pt.2

5. Conclusion

In conclusion, threads and processes are both methods of concurrently executing code on a computer. Threads are a way of achieving concurrency within a single process, while processes are independent units of execution that can be run concurrently with other processes.

There are several key differences between threads and processes:

- Threads share memory space, while processes have their own separate memory space. This means that threads can access and modify the same data within a process, while processes must communicate with each other through some kind of interprocess communication mechanism in order to share data.
- Threads are generally lighter-weight than processes, as they require less overhead to create and manage. This makes them more efficient for certain types of tasks, such as those that involve frequent communication or synchronization between threads.
- Processes are generally more isolated than threads, as they have their own separate memory space and cannot directly access the memory of other processes. This makes them more suitable for tasks that require a high degree of security or isolation.

Overall, the choice between using threads or processes will depend on the specific requirements of the task at hand. Both have their own strengths and weaknesses, and the most appropriate choice will depend on the needs of the application.