

**Benchmarking scripts**

for multi

-

core CPUs



**By Deac Dan Cristian**

**Content**

1. **Introduction** ................................................................................ 1
2. **Bibliographic research** ...............................................................1
   1. Clock() .............................................................................1
   2. Threads ...........................................................................1
3. **Analisys** ………....…………..………………………………………..2
   1. Make the scripts in C/C++ …………………………………2
   2. User interface ……………………………………………….2
4. **Design** ……………………….….……………………………………..3
5. **Implementation**……………………………………………………6
6. **Conclusions**……………………………………………………………8
7. **Bibliography**……………………………………………………………8

# Introduction

This project is a benchmarking application that performs several measurements on multicore processors. The project will be done on a linux machine using the available tools. The

development of this project as well as this document is achieved using open source software. The script is intended to work in any operating system and will be available on github for

download.

**>git clone https://....**

Compilation(Linux):

**>g++ script.cpp** –**o executable -- for 3 executables**

**>chmod +x executable**

**>python interface.py**

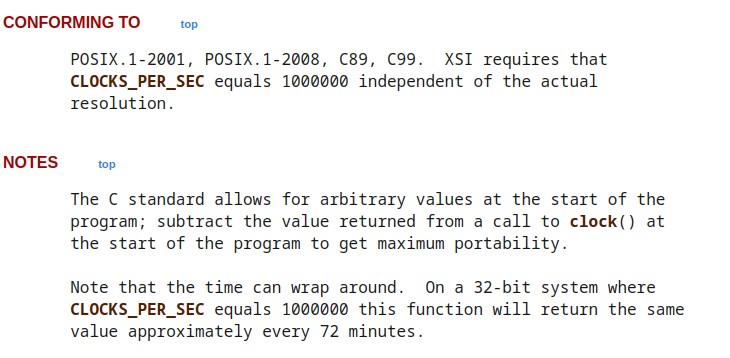
* Possible needed library : libssl\_dev for Debian systems and something similar for the openssl lib for other distros.

# Bibliographic research

2.1 Looking in the man7 proves to be very useful understanding the libraries and the use

of functions that attempt to measure the performance of a CPU. One example is the time.h which enables the use of the clock() function.

Linux Manual page: <https://man7.org/linux/man-pages/man3/clock.3.html>



For research I also used the intel documentation for using inline assembly to micro benchmark the code. Then I used a bunch of YouTube tutorials to learn python UI with pyqt5

Then I used the ideas from laboratory 2 to perform a stress\_test function that performs addition and

XOR in assembly and uses the chrono library to benchmark the results.

For the Encryption test I used the prior knowledge from the OS classes to encrypt passwords

And split this task for multiple threads equally if possible so it makes sense to use a multithreaded

application

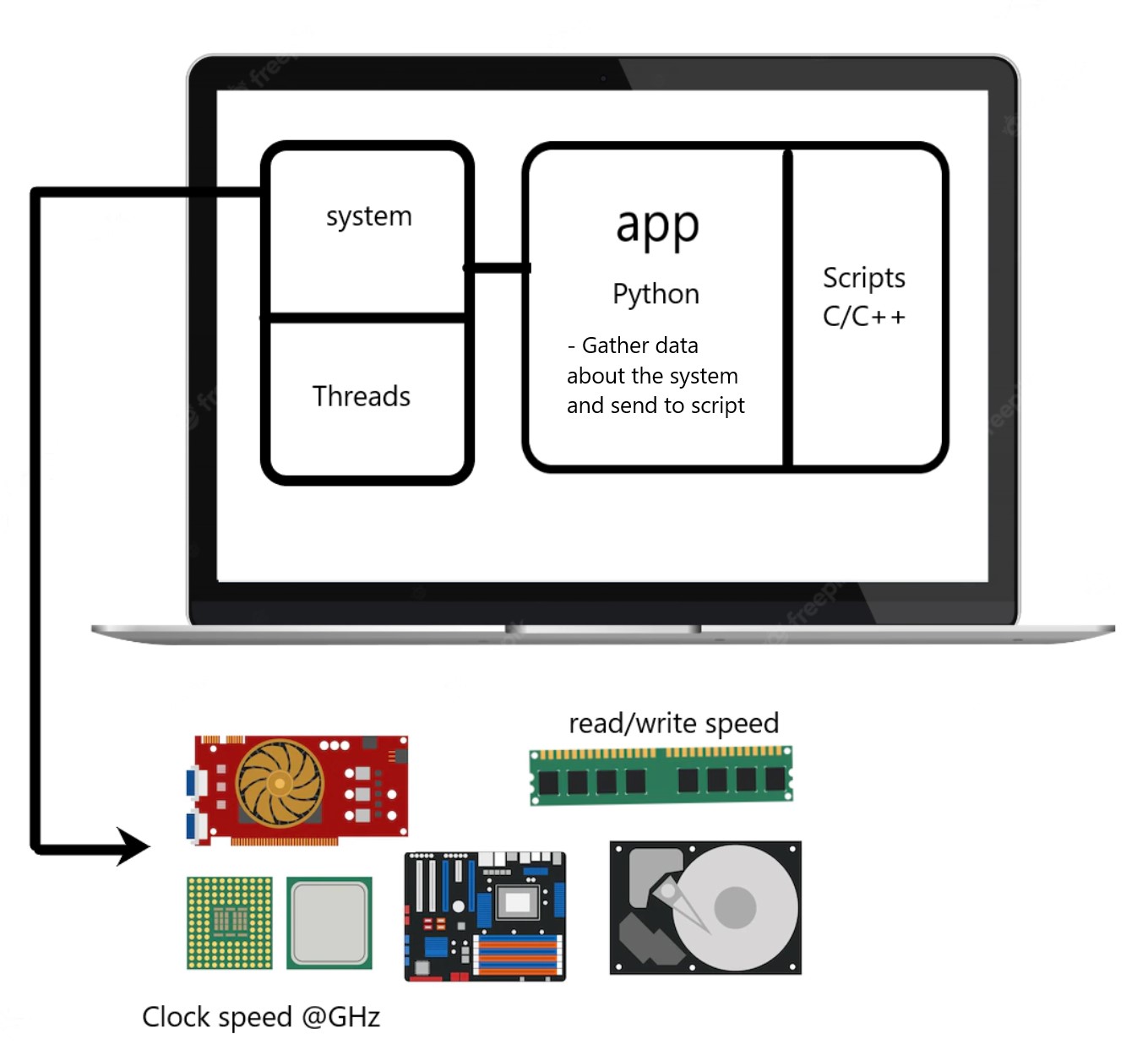
## Analysis

The first part of the project will be focused on the development of the functional part. The core should be based on several C/C++ scripts that are executed by the user interface, or their results are stored in several files. Then the files are interpreted by the user interface and shown with some kind of graphical representation.

The interface should run the script with certain arguments that are read by the front end of the application. A library will be used to gather this information and then the user interface will wait for the script to run, and then display the results. An other solution would be to generate the results in pdf files and open them automatically in the user interface after the scripts were run successfully.

The second part will be the development of the user interface. I would prefer to use C# for windows, but if this project will be developed with linux in mind I am thinking of switching on something python based with tkinter library.

## Design



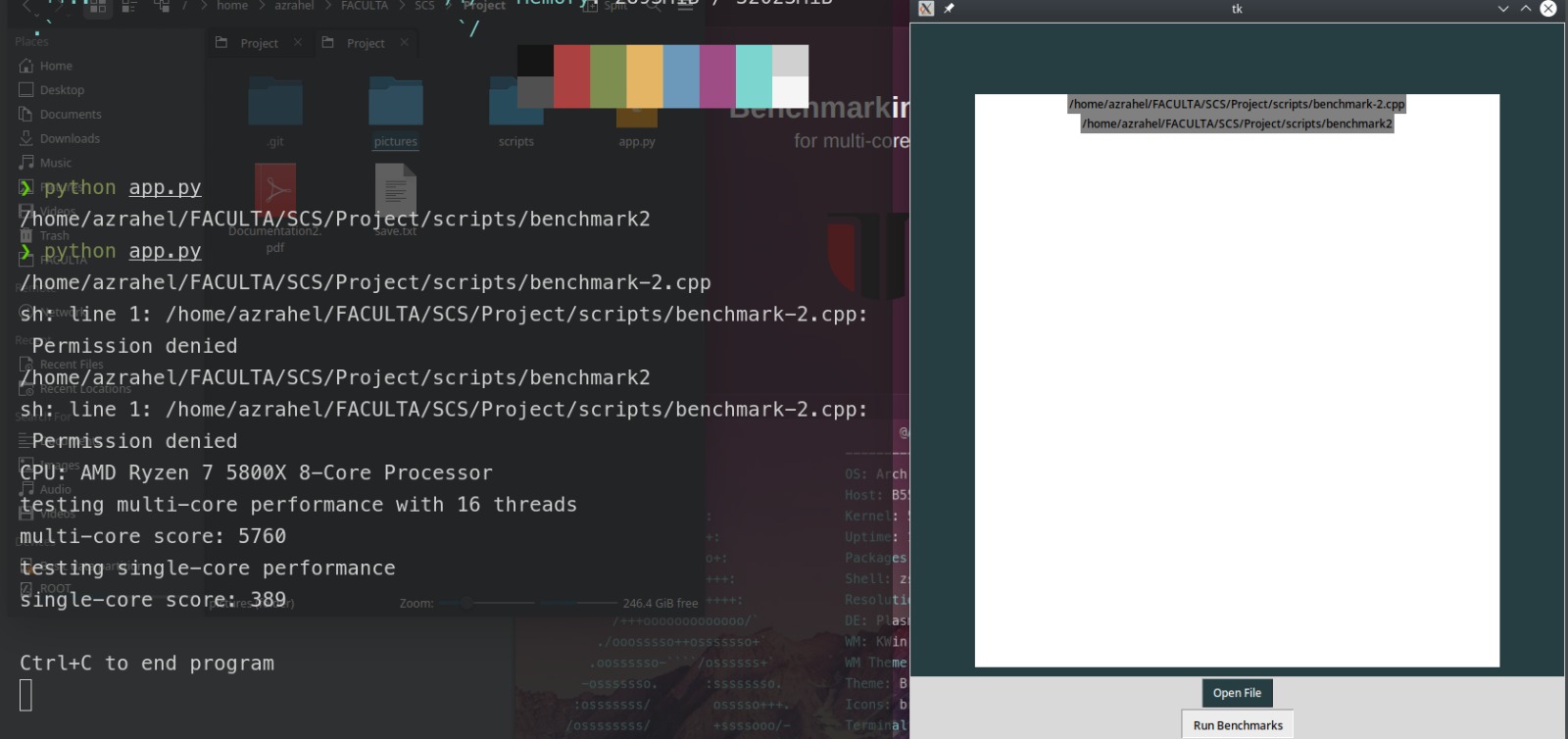
**Figure 1 – design**

Application Interface:

* Should be able to select which tests to be ran
* Start benchmark / Stop benchmark
* Open results button (which could open a generated pdf file by the script)

Script

* Get data from arguments (probably only used for displaying it in the chart)
* Run Test that was given by the argument - Put the results in a chart – pdf file



**Figure 2 - UI**

This version of the app is based on tkinter framework and is able to open a folder and select from the system the desired benchmarks that can be ran. In the future version I am thinking of switching completely to PyQt5. It offers a broader range of tool for UI design.

Functionality wise, it uses right now a default “stress\_test” that is a prewritten function which tests the performance of the single core and the multicores. Then it times them using functions from the chrono library and prints the “score” in the terminal in this state.

The implementation uses some assembly code to check for a certain place in memory to get the brand of the CPU and other useful information. That is the only things so far that runs on assembly code.

I changed from tkinter to pyqt5 in order to use the QT designer and for a simpler code

Graphical user interface

Description automatically generated

The graph is generated using the gnuplot library. Basically, I run for a specific number of threads a certain test, either the stress\_test or the encryption to SHA1, and I send the thread execution time for each thread number to a text file called “data.txt”. Then I generate the plot using this data. The contents of the data.txt can be seen in the terminal opened as a result of pressing Run.

## implementation

void stress\_cpu(int operations)

{

#define more asm("addl $1, %eax\nxorl $1, %eax\n");

#define Much\_more more more more more more

#define MMuch\_more Much\_more Much\_more Much\_more Much\_more Much\_more

#define M4 MMuch\_more MMuch\_more MMuch\_more MMuch\_more MMuch\_more

#define M5 M4 M4 M4 M4 M4

#define bunch\_of\_asm M5 M5 M5 M5 M5

asm("movl $1, %eax\n");

for (int i = 0; i < operations; i++)

{

bunch\_of\_asm

bunch\_of\_asm

}

}

void \*encrypt\_passwords(void \*threadarg) {

struct thread\_data \*data;

data = (struct thread\_data \*) threadarg;

int i, start, end, thread\_id, num\_threads;

//double start\_time, end\_time;

thread\_id = data->thread\_id;

num\_threads = data->num\_threads;

// Each thread encrypts a portion of the plaintexts array

start = thread\_id \* NUM\_PASSWORDS / num\_threads;

end = (thread\_id + 1) \* NUM\_PASSWORDS / num\_threads;

clock\_t start\_time = clock();

for (i = start; i < end; i++) {

SHA1(data->plaintexts[i], strlen(data->plaintexts[i]), data->encrypted[i]);

}

//end\_time = omp\_get\_wtime();

clock\_t end\_time = clock();

data->execution\_times[thread\_id] = end\_time - start\_time;

//usleep(1000);

pthread\_exit(NULL);

}

This code defines a function called "stress\_cpu" that takes an integer argument "operations" and uses inline assembly to stress the CPU by performing a large number of operations. The function begins by defining several macros (**more, much\_more**, etc.) that each expand to a sequence of assembly instructions. Each macro consists of a series of other macros (**more**, **much\_more**, etc.) that are recursively defined in a similar manner. The first instruction in the macro **more** is "addl $1, %eax\nxorl $1, %eax\n", which adds 1 to the value in register %eax and then performs a bitwise exclusive OR operation on the result, also storing the result in %eax. The macro **bunc\_of\_asm** expands to a sequence of 5 M5 macro expansions, M5 in turn expands to a sequence of 5 M4 macro expansions and so on. The following inline assembly instruction "movl $1, %eax\n" move the value 1 to %eax. Then the function uses a for loop to iterate "operations" times, in each iteration it performs the macro M6 twice, which ultimately results in performing all the assembly instructions defined in all the macros. The net effect of this code is to perform a large number of simple operations on the CPU, in order to stress it and potentially detect any errors or instability in the system.

The code above is where we test each thread ‘s duration. It uses time.h clock() function for calculating the end time and start time and then saves the execution time to the data structure.

## Testing and Validation

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

## Conclusions

This application is a simple benchmarking script that pretty much introduced me into using inline assembly and some more C code and python code. It does the simple job of checking a computer’s cpu specifications and then can perform a stress test on it and outputs a score for the single and multicore performance.

This assignment gave me an insight into the world of multicore processors and I will be able to better understand what Intel or AMD say during their new CPU showcases and release livestreams.

## Bibliography

[PyQt5 Tutorial - Setup and a Basic GUI Application - YouTube](https://www.youtube.com/watch?v=Vde5SH8e1OQ)

SCS – lab 2

Operating systems last year laboratories where we learnt about threads