HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2017
EXAMINATION MINIPROJECT

1. Instructions

Your task is to implement and optimize a small decryption program. You are given an incomplete unoptimized code and instructions for how to complete it.

The program takes three input arguments: filenames of the input and output file, and the number of threads to use. The initial code you get reads the input file into a buffer and writes the final result to the output file, but the actual decryption needs to be written.

In addition to the decryption process, the program also computes a number called `characteristicNumber` – see the main() function in `decrypt.c`. Unoptimized code for this computation is provided in `funcs.c`.

The decryption that is to be implemented consists of the following 5 steps:

- Step 1: Call `ShuffleBitsA()` for each byte in the buffer
- Step 2: Call `ShuffleBitsB()` for each byte in the buffer
- Step 3: Swap bytes pairwise in the buffer
- Step 4: Call `ShuffleBitsC()` for each byte in the buffer
- Step 5: Reverse order of bytes in the buffer

In step 3, the program should swap bytes pairwise in the buffer; if the total number of bytes is odd, the last byte remains unchanged in step 3.

Unoptimized code for the functions `ShuffleBitsA`, `ShuffleBitsB` and `ShuffleBitsC` are provided in the source file `utils.c`.

A few encrypted files are provided; test your decryption code by running it on those files, and check that the decrypted files are OK.

Use all your knowledge from the course to make the computation and decryption code run as fast as you can; this includes both code changes and changing compiler options in the Makefile. You can choose to use either Pthreads or OpenMP for parallelization, using the number of threads that was specified as input to the program.

You are free to use any computer you want for this miniproject, you can run on the lab computers at the university, or on your own laptop, or some other computer. However, your code must anyway be portable, you should verify that it can be compiled and executed on either the lab computers or the university's Linux servers, e.g. `fredholm.it.uu.se`.

If you want to do some optimizations that mean the code will not run on the lab computers or `fredholm.it.uu.se`, then you should include that code as an option, the code you submit should by default work as a portable code. Then you

---

Date: March 2, 2017.

can describe in your report how to enable your non-portable optimizations, and describe what the hardware requirements are.

As you work on optimizations, remember to check that the results are still correct. For the decryption, check that the decrypted files are OK. For the computation of `characteristicNumber`, check that the computed value does not deviate too much from what you get with the original, unoptimized code. The absolute value of the *relative error* for `characteristicNumber` should not be larger than 10^{-9} for the given image files. (If the original value is x_0 and your computed value is x_1, then the relative error is defined as $\frac{x_1 - x_0}{x_0}$)

1.1. **Part 1: implement decryption and verify that it works correctly, without worrying about optimizations.** Implement each of the 5 steps described above by making appropriate changes in `decrypt.c`. The places in the code where changes are needed are marked with "`TODO`" comments. To verify that your code works correctly for the smallest case, the encrypted file `abcdefgh.txt`, you can use the function `PrintBufferContents` before and after each step. Then you should get the following results:

```
buf0: 199 13 197 135 71 133 69 7
buf1: 203 14 202 75 139 74 138 11
buf2: 173 104 45 172 169 44 41 168
buf3: 104 173 172 45 44 169 168 41
buf4: 104 103 102 101 100 99 98 97
buf5: 97 98 99 100 101 102 103 104
```

Note that `buf0` here means the data from the input file (before step 1), `buf1` means the data after step 1, `buf2` means the data after step 2, and so on, as in `decrypt.c`.

When the smallest case works, check also that you can decrypt the three image files. Use any image viewing program to look at each image after decrypting it. If you want to test your code on some larger file, there is an encrypted mp3 file of about 160MB that you can download from the "Encrypted files for miniproject" document collection in the Student Portal. After decrypting it, you should be able to listen to the music by opening the decrypted file using any music player program that supports the mp3 format.

1.2. **Part 2: optimize everything you can.** Optimize everything you can, including making changes in `funcs.c` and `utils.c` as well as in the main program in `decrypt.c`. Parallelize using either Pthreads or OpenMP. You are free to change the structure of the program in any way you want, as long as the final results are still correct. Note that correct results mean both that the resulting decrypted file is correct, and that the computed `characteristicNumber` is correct.

Focus on achieving the best possible performance for large input files.

For Part 2, remember to *save different working versions of your code so that you can compare different versions to see the effect of your optimizations.*

1.3. **Report.** Write a short summary, 2-3 pages, discussing the performance of your code, showing the effectiveness of your optimizations and your parallelization. If an attempted optimization does not work and you know why it does not work, include this in your report. *Hint: figures and tables are good ways of presenting results concisely.*

Note: max 3 pages for the report! It is important to be able to present results consisely, show us that you can do that. Reports longer than 3 pages will not be accepted.

1.4. **Format of submission.** Your submission should consist of a single compressed tarball named `miniproj.tar.gz` including your code and report files:

- Runnable code for part 1 in a directory called `part1`
- Runnable code for part 2 in a directory called `part2`
- Report in pdf format, maximum 3 pages long, filename `report.pdf`

For part 2, the code you submit should be your best code with the best compiler options in the makefile. We should be able to test your code by unpacking your submitted `miniproj.tar.gz` file and going into your `part1` or `part2` directory and simply doing "make" and then running the program.

You should verify that your submission has the correct form in the following way: create a new directory and copy your `miniproj.tar.gz` file and the encrypted `image1.jpg` file into that directory. Then cd into that directory so that if you do "ls" you see only those two files listed. Then type *exactly* the following commands:

```
tar -xzf miniproj.tar.gz
cd miniproj
cd part2
make
./decrypt ../../image1.jpg image1_decr.jpg 2
```

Then your optimized code should run and decrypt the `image1.jpg` file, using 2 threads. If doing the same thing for the part1 directory, that should give your unoptimized code.

Note: your submission must follow exactly the requested format. Part of our checking of your submissions will be done using a script that automatically unpacks your file, builds your code, and runs it for some test cases, checking both result accuracy and performance. For this to work, it is necessary that your submission has precisely the requested form. Verify that your submission follows the requested format by testing it as described above.

If you want to demonstrate different versions of your code, you can include separate directories for other versions, for example as "part2other", but the `part2` directory should anyway contain your final result, your best code.

2. Submission and Examination

The exam will take place during three days, March 15-16-17, in room *2507*.

On the days of the exam, you will be interviewed in turn, by the computer. This is your opportunity to show us what you have learned! We will ask questions about this miniproject and your code, as well as some other questions about the course contents.

If you want to take the exam, we ask you to submit the code and report under "Examinatory miniproject" in the Student Portal by the *10th of March, 23:59*. After submitting your miniproject you will get to pick a time slot for the oral examination, some time 15-16-17 March. The Student Portal will be used for that, students get to choose time slots on a "first come, first served" basis.

Important: the exam work should be carried out *individually*. You can talk to other students, but don't do it in front of a computer and don't write anything

down. The bottom line is that we have to be convinced that the handed-in code and report are done by you and understood by you.

Also important: Bring an ID card, passport or equivalent, on the exam day so that we can verify that you are who you say you are.

3. Grading

You will be graded according to a (secret) rubric based on the complete course contents. Both your miniproject submission and your answers to questions (both specific questions about this miniproject and questions about other course contents) will be taken into account. The grade requirements are as follows:

- **3** You must demonstrate an understanding of the fundamentals of the course. Your code works correctly and you have made a reasonable attempt at optimizing and parallelizing the code.
- **4** You have exhaustively tried to optimize everything, and you're able to reason about the performance of your code at a high level and know why certain optimizations worked/didn't work.
- **5** To earn the highest grade, you must also formulate a new function/routine that allows you to demonstrate at least one optimization technique that you were not able to use (i.e. benefit from) in your final solution. In other words, develop a little code that performs some kind of computation that you invent and devise it in such a way that you can showcase an optimization technique not seen elsewhere in your code. Put this in a separate file, e.g. `extra.c`, and briefly describe it in your report.

4. Questions

If there are any questions, email: `elias.rudberg@it.uu.se`