Lab Session 1: Programming a shell tool
24296 - Sistemes Operatius

1 Delivery exercise:

You will implement a command-line utility that will serve to both generate and verify the CRC code of a file, as seen in the seminars. The different behaviours will be controlled by the given arguments. The code is in the files crc.c and crc.h. You will have to use the function "crcSlow", which has a similar signature as seen in class. Notice that its return is of the type unsigned short, which occupies 2 bytes, instead of 1, as in the seminars!

The first argument must be a path to an existing file. The filename of the CRC file will be the same as the data file, with *.crc* added as a suffix, and when the program option is to generate the CRC, it should be created if it does not exist, or overwritten if it already exists. When the program is in its verify modus, the CRC file should be open in a read only mode.

The rest of the arguments can be, in any order:

- *-verify* The program will be read the CRC file, and compare its content to the recomputed CRC. It will write in the standard output "file OK" or "file has errors", depending on whether or not are errors. If the *-maxNumErrors* option is active, then it will output "file OK" if the number of errors is below the threshold.
- *-generate* If this option is active, the CRC file will be created, or overwritten if it exists .
- *-maxNumErrors numErrors* this option is only correct when the -verify option is also active. It indicates the maximum number of errors before declaring that the is file is incorrect. You can assume that numErrors will always be a correct integer.

If the arguments are incorrect, exit with the error code 22. If any of the files do not exist, or cannot be created, exit with error code 5. Otherwise, the program will do the specified functionality.

Lastly, you will profile your code, and accumulate in a counter the amount of time spent while doing the different read operations, and in another counter the time spent in the computation of the CRC. For this, you can use the functions given in the files timer.c and timer.h; that return the elapsed time, in microseconds, between the calls to "startTimer" and "endTimer". Here is a code snippet showing its usage:

```
long totalTime_red = 0; // Initialise the timer
...
    startTimer();
    bytesRead(fd, buff, 256);
    totalTime_read += endTimer();
...

printf("Microseconds spend in read are %d\n", totalTime_read);
printf("Microseconds spend in computing the CRC are %d\n", totalTime_crc);
```

1.1 Hints

1. As the arguments can be given in any order, this requires to iterate over each argument, and compare them to the list of options. You can find an example of this way of processing arguments in "corrupt.c", a program that randomly changes some bytes of a file.
2. Remember (or search) the use of the following string processing functions: *strcmp*, *strcat*, *atoi*.
3. Now the files can be non multiples of 256. You have to handle the last block in a different manner.

1.2 Possible usages

Valid calls:

- checksum a.txt -generate
- checksum a.txt -verify

- checksum a.txt -verify -maxNumErrors 5
- checksum a.txt -maxNumErrors 5 -verify

Invalid calls:

- checksum -generate a.txt
- checksum a.txt -generate -verify
- checksum a.txt -generate -maxNumErrors 5

2 Theoretical questions

Answer these questions in a brief report.

Theoretical questions

1. Explain the parsing of the arguments, and the checks that you have implemented to verify that they are correct.
2. Suppose that you have a variable blocksize, that is given per command line. What would you need to modify?
3. Explain the handling of the last block, which might be smaller than the blocksize.
4. Compute the total timing for computing the CRC, and reading, of the file "test_large.txt". Assuming that you your computer only has a single CPU, and knowing that the CRC does not use any I/O operation, discuss about the potential acceleration of using concurrency on this program.

3 Submission instructions

Include a zip file with the following:

1. The C code of the shell tool whose behaviour has been specified in the previous section.
2. All the other C code (the code and header for computing the CRC, any utilities you might use).
3. A bash file with the build instructions to compile the C files and generate a binary called *checksum*.
4. A pdf with the theoretical questions.

Remember that the code should compile and execute with the bash command you include .