Instituto Superior de Engenharia de Lisboa

BEng in Computer Science and Engineering

System Virtualization Techniques, Autumn/Winter 2023/2024

First coursework assignment

Unit test frameworks for code written in C face an additional challenge, when compared with equivalent frameworks for other environments, such as JUnit (JVM), NUnit (.NET) or Mocha (JavaScript), as in the native environment there is the possibility that the code under test might execute some illegal action, causing the test process to be immediately terminated by the operating system. To deal with this, some unit test frameworks for C/C++ code run the central test control and reporting code in one main process, while executing each specific test in separate dedicated processes.

In this assignment, you will develop CHUTA (***CH**elas **U**nit **T**est **A**pparatus*), a unit test framework for code written in C that *kicks* off a new dedicated process for each test it runs. A unit test is written as a function that *kicks* around specific functionality of some project's code in order to evaluate its correction and sturdiness.

1. Create a dynamic link library `libchuta.so` with the main test control function `run_function_tests`. This function takes in an array of pointers to test functions and, for each of those functions, it creates a new process and invokes the test function in it. If the test function returns, the test is considered successful and the exit status of the dedicated test process will be EXIT_SUCCESS. If some assertion fails during the test, the dedicated process must exit immediately with EXIT_FAILURE. If the test execution is terminated by a signal (usually due to some serious error), the main test control process can use WIFSIGNALED and WTERMSIG on the exit code of the test process to get information about it.
   For each test function, `run_function_tests` prints a line to standard output with the number of the test (its index in the array of test functions) and the result of the test: SUCCESS, ASSERTION_FAILED or TERMINATED(*signum*) – where *signum* is the number of the signal that terminated the test process. If `stop_at_first_failure` is `true`, no more tests are executed after one of them fails. In the end, a summary line is printed with the total number of tests, the number of tests executed, and how many of them succeeded or failed.
   The library is accompanied by a header file `chuta.h` with its public interface and auxiliary definitions, including the `CH_ASSERT` macro. This macro evaluates `test_expression` and, if it evaluates to `false`, causes the test process to immediately report a failure.

   ```
   // Auxiliar definitions in chuta.h

   #define CH_ASSERT(test_expression) \
      do { if (!(test_expression)) { /* (to be defined by you) */ } } while(0)

   typedef void (*test_function)();

   void run_function_tests(test_function tests[], size_t num_tests,
                           bool stop_at_first_failure);
   ```

   Tag this version on the GitHub repository with: **`CW1-1`**

2. Improve the library from step 1, allowing a test failure to report more information to the main control process. In this version, if an assertion fails, the following information is written to standard error in the test process: the expression that failed as a string; an optional message, if the assertion uses a new macro `CH_ASSERT_MSG(test_expression, error_msg)` with an explicit error message; and the file and line number where the assertion failed. The main test control function will capture these text lines and write them to its own standard output after the test result report line. Also find a (simple) way to include the test function name in every report line written by `run_function_tests`.

   Tag this version on the GitHub repository with: **`CW1-2`**

3. The library should also support testing programs (executable files) that operate by reading text from standard input and writing text to standard output (e.g. `cat`, `grep`, `sort`, `wc`). For that purpose, add a new function, `run_stdio_program_tests`, to run an array of such tests. Each test is specified by four arguments: a test name, a string with the command to run the program (including arguments); the path to a text file that will serve as standard input to the test; and the path to a text file with the expected output. Run each program with the specified input and check that the output matches, then print a report line, similar to the ones written by `run_function_tests`.

   Tag this version on the GitHub repository with: **`CW1-3`**

At every step, deliver:

- Source code for `libchuta.so`
- Its include file `chuta.h`
- One or more files with example code and unit tests, that demonstrate all the functionality of the library.
- An entry `run_tests.c` file with the `main` function, that runs all available demonstration tests by invoking `run_function_tests` (and `run_stdio_program_tests` for step 3).
- At least one `Makefile` in the base directory to build the whole project.

Do not submit binaries and other unneeded files to the repository.

For the absolute final version, use the *tag* **`CW1-DONE`** on the GitHub repository.

ISEL, September 23nd, 2023

Submission last date: October 8th, 2023