This document includes the guidelines for developing the programming exercises for the Operating System course (CIIC 4050 / ICOM 5007).

1. File structure

The project needs to have the following file structure:

```
project/
├── CMakeLists.txt
├── include/
│   └── functions.h
├── src/
│   ├── functions.c
│   └── main.c
└── build
```

Where:

- The include folder contains all the header files with the prototype of the functions and macro definitions.
- The src folder contains all the source code files including the main file and the necessary files for the function definitions
- The build folder will contain all the files resulting from the compilation process, including object files and the executable file for the program.
- The CMakeLists.txt is a **mandatory file** that contains the necessary steps to build the project using cmake. It will be used for standardization purposes.

2. Coding Style

All developed codes must be compliant with the Google style guidelines, which are highlighted in another document. This style document includes guidelines regarding header *files, naming,* and *formatting*.

On the other hand, functions must be documented using the Doxygen format to define their primary task and the input and output parameters. Doxygen is a documentation generation tool commonly used for C, C++, and other programming languages. It uses specially formatted comments within the source code, known as Doxygen comments, to generate comprehensive documentation in various formats, such as HTML, PDF, and LaTeX. The objective of the Doxygen format is to provide a standardized way to document code, making it easier for developers to understand, maintain, and collaborate on projects. By embedding the documentation directly within the codebase, Doxygen ensures that it stays closely aligned with the source code, helping keep it up-to-date and accurate as the code evolves. An example of a Doxygen documentation for a function that add two integers is presented next:

```
/**
 * @brief Adds two integers and returns the result.
 *
 * This function takes two integer parameters, adds them together,
 * and returns the resulting sum.
```

```
 *
 * @param[in] a The first integer to add.
 * @param[in] b The second integer to add.
 * @return The sum of the two integers.
 *
 * @note This function does not handle overflow.
 *
 * @example
 * int result = add(3, 4);
 * // result now holds the value 7
 */

int add(int a, int b) {
    return a + b;
}
```

Of all these aspects, the formatting can be automatized using software tools. A .clang-format file is a configuration file used by the Clang-Format tool to format code according to specified style rules automatically. This file allows developers to define consistent code formatting conventions, such as indentation, spacing, and brace placement across a project or team. By placing a .clang-format file in a project's directory, Clang-Format can apply the defined style to all code files within that directory, ensuring uniformity and improving code readability. The .clang-format file is highly customizable, allowing developers to establish the formatting rules to match their organization's preferred coding standards or guidelines.

In this case, the tool *clang-format* can be used within the editor VSCode to apply the Google coding format automatically. To achieve this, you must:

1. Copy the `.clang-format` file to the folder **containing** your working directory. **It's not the project folder, but one level above.**
2. Go to `settings->formatting` and enable `"format on save"`
3. On `"format on save mode,"` choose `"file".`

    a. **Set the tab key to use "two spaces"**

    Click on the bottom right bar where it says "spaces" and set its value to 2.

3. Compile the project

The course projects will be compiled using a `CMakeLists.txt` file. This file is a script used by CMake, a cross-platform build system generator, to configure the build process for a software project. It defines the project's structure, specifies the source files, sets compiler options, handles dependencies, and can manage platform-specific configurations.

The CMakeLists.txt file is essential because it enables developers to generate build files (like Makefiles on Linux or Visual Studio projects on Windows) in a way that is consistent across different environments. This makes compiling and maintaining the project on various platforms easier, ensuring the build process is flexible and repeatable.

Additionally, CMake simplifies the management of complex projects by automating the detection of libraries, setting up include paths, and handling compiler and linker flags, thus reducing the manual configuration effort.

a. Example of project compilation using CMakeLists.txt

- Create a project using the file structure of section 1. In the `include` folder create the file `hello.h` and in the `src` folder create the files `main.c` and `hello.h`.
- In the `hello.h` file put the following code:

```
#ifndef HELLO_H
#define HELLO_H

void PrintHelloWorld();

#endif  // HELLO_H
```

- In the `hello.c` file put the following code:

```
#include "hello.h"
#include <stdio.h>

void PrintHelloWorld() { printf("Hello, World!\n"); }
```

- Finally, in the `main.c` file put the following code:

```
#include "hello.h"

int main() {
  PrintHelloWorld();
  return 0;
}
```

- Now, create a `CMakeLists.txt` file in the root folder of the project with the following code:

```
cmake_minimum_required(VERSION 3.10)

# Set the compiling mode flags
set(CMAKE_CXX_FLAGS_DEBUG "-g -O0 -DDEBUG")
set(CMAKE_CXX_FLAGS_RELEASE "-O3 -DNDEBUG -s")

# Set the project name
project(HelloWorld)
```

```
# Include the current directory for header files
include_directories(include)

# Add the executable
add_executable(hello_world
    src/main.c
    src/hello.c)
```

- Generate the build files from a terminal by getting into the `build` and using cmake:

```
cd build
cmake ..
```

- Now compile the project:

```
Make
```

- Finally, run the program:

```
./hello_world
```

4. Compile using CMake in VSCode

- Install the C/C++ Extension Pack in VSCode.
- Open the folder from the exercise in VS code.
- Make sure the root folder contains the CMakeLists.txt file.
- Go to the cmake extension mode on the left bar:
    - Select project folder
    - In configure select GCC compiler and then clic on the 'sheet icon' to configure
    - Select debug or release mode
    - in Build, click the build button

- Now you can run the program or you can debug the program (if compiled in debug mode). To debug, you need to set a breakpoint first, then click on debug. To run, you only need to click on the Launch button.