Universidad Zaragoza 1542

Memory Leaks

Fundamentals on Computing for Robotics, Graphics and Computer Vision

Prerequisites

As in previous lab, his lab relies on the linux machines from the DIIS deparment. Either a local machine from the laboratories, our you can directly connect to `lab000.cps.unizar.es` machine, which has the same linux machine that the lab machines. So the single actual prerequisite is to be present in the lab or have an internet-conected computer. Please remember that your home directory is the same in all the machines of the department, so you are free to stop and continue working on a different machine.

Setup steps

Initially, you have to log into a machine directly or though ssh with `ssh lab000.cps.unizar.es` if you connect to lab000. Don't forget to add the flag –Y to have X Windows support. Supporting X Windows on your machine may require the installation of Windows Subsystem for Linux or XQuartz for Windows 10/11 or macOS, respectively.

Once you have logged in, we are going to clone the repo and build the binaries to start debugging them. So these are the required commands:

```
cd <your root directory for FoC repo>
git clone git clone https://github.com/adolfomunoz/FoC.git
cd FoC
```

In case you already cloned the repo, please update it to the latest version with the following commands:

```
git pull --ff
```

If you have changes that you want to preserve, please check git stash.

Once the clonation or the upgrade have finished, let's continue building in *debug mode* the assigment's programs:

```
cd src/assignment_12_classes_materials
mkdir build-debug
cd build-debug
cmake -DCMAKE_BUILD_TYPE=Release ../ # generate the Makefile with cmake
make VERBOSE=1 -j4 # compile the examples in parallel with 4 jobs
```

This assigment includes new programs that you have to write. To work with them, please add them into the `CMakeLists.txt` file of the assigment.

Initial questions

1. What is the effect of `VERBOSE=1` in the output of the `make` step?
2. With `make`'s output, could you verify the requirements to use valgrind?

Finding Memory Leaks with Valgrind

Valgrind is a very powerful to to find memory leaks. In the first example, your job is to build and run the `leak_factorial` application within Valgrind and find the memory leaks.

1. In which lines there are memory leaks? Please concisely explain the issues, if any.
2. Solve the memory leaks with smart pointers in the new `correct_factorial.cpp` file, that your submission should include.
3. Verify with Valgrind that `correct_factorial` does not have any memory leak.
4. Please repeat the previous 3 questions for the `find_memory_leaks.cpp` source file.

Finding other errors with Valgrind

Valgrind helps to find other errors as well. For example, you may remember double deletion of dynamically allocated memory.

Please read the source code of `incorrect_delete.cpp` program.

1. What is the memory related error?

Now, build and run the application in debug mode with valgrind and answer the following questions.

1. Does valgrind find the problem?
2. How would you solve it?

For the last step, please submit a `fixed_incorrect_delete.cpp` version of the file that automatically handles the release of the memory.

Circular dependencies

One hard to spot and hard to fix issue dealing with pointers is circular depencencies that occurs when at least two objects have a chain of pointers among them, so they cannot be deallocated.

The source file `circular_dependency.cpp` includes includes a circular dependency between two nodes.

1. First, please build and run the `circular_dependency` program and verify whether all required destructors are fired, if there is a memory leak? If yes, please identify the non-released objects.
2. Could you identify which class is causing the trouble, if any?
3. Read the documentation of std::shared_ptr to check how C++ provides support to solve circular dependencies.
4. Write a fixed version of `fixed_circular_dependency.cpp` to include it in your submission.

[Optional] Use AddressSanitizer

The `CMakeLists.txt` includes a `buffer_overflow` target that links the application with the Address Sanitizer. Depending on your target system, you may be able to run the application and observe the output.

Submission

Please group the all the files within a zip. In those assignments with text questions, add then at the beginning of the file as C++ comments.

Before submitting, remember to talk to your teacher and have a **small interview** describing and answering questions about the job you have done and the decissions you have made. This interview is *20%* of the total grade of the assignment, while the submitted material is evaluated for the other *80%*.

All source code files, must include a comment on top with the names and NIAs of the students involved on their development. Then, all source code files, including directory structure, must be compressed into a single `zip` file with the following naming:

- **`memleaks_<nia1>_<nia2>.zip`**, where `<nia1>` and `<nia2>` are the NIAs of the involved students, if the work has been done in pairs. In this case, **only one** of the students must submit the work in Moodle.
- **`memleaks_<nia>.zip`**, where `<nia>` is the NIA of the involved student if the work has been done individually.

The compressed file must not contain anything else besides the source code files. Particularly, do not submit any binary file, neither executable, library nor object. The submission of the `zip` file must be done through the corresponding task in Moodle before the stablished deadline.

Comments with ideas, improvements, suggestions,… are always welcome.