SEPM/Linux Tasks Winter Term 2009

SE/Linux Team*

October 12, 2009

1 Task 1

1.1 Focus

The focus of the first task is to introduce you to your new development environment as gently as possible and as roughly as needed :) You will get a test and a development environment and a specification of what we expect you to implement in C++. More important than your code will be setting up the build system and understanding what you are expected to do.

1.2 Mission

Write a simple command-line client for the EHR system used by the either the patient, the physician or the pharmacist. The user can display a list of prescriptions in the patient's EHR and select one for PDF generation.

1.3 Documentation

Documentation on how to use free software has one big problem: it is scattered all over the net and no one is paid for writing it, which means you have to spend some time searching. Google is your friend.

One of the best and most accurate sources of information are the man and info pages. They are present on every (good) UNIX installation but reading and, more importantly, understanding them needs some practice. Be sure to try to use them. Use `apropos` and `whereis` to locate man pages on the system and `man` or `info` to read them. Of course `man man` and `info info` will provide some help.

*se-linux@inso.tuwien.ac.at

1.4 Tasks

1.4.1 Reading

- Read (the relevant part of) the Ice documentation on [2], including, but not limited to:
  - Ice overview
  - A hello world application
  - The slice language
  - Slice for a simple file system
  - Client side slice-to-C++ mapping
  - Ice Properties and Configuration
  - Using IceSSL
  - Streaming Interface
- Read up on C++ [4, 5] or, if you are familiar with Java, a very short tutorial about transitioning from Java to C++ [3].

1.4.2 Designing and Coding

Create a C++ application using the given interface and use the running server.

Coding guidelines Following a good, well-written coding standard makes your code easier to read and to maintain, especially if you are new to C++ programming. Various coding standards and guidelines für C++ exist, many of them contradicting each other. Please find a set of standards that suits you and follow it faithfully. If you are, for example, familiar with Java and its coding conventions [6] you can use them almost verbatim for C++. Nonetheless we recommend the style guide published in [1] as it is very concise and well-thought out. It is also the preferred style of the SE/Linux team.

Coding rules Please follow the rules and suggestions discussed in the C++ lecture. Write a C++ program using all the strengths and conveniences C++ has to offer. Don't use old C-style functions like `strcmp`, `printf`, `atoi` and the like. Perfectly safe, well-working and more usable C++ classes and methods exist to accomplish anything any of the old C Standard Library functions could do. If in doubt how to implement or use a particular feature or how to accomplish a task like converting a string to a number or vice versa ask a tutor or use your Google-Fu.

Do not mix C with C++ code. If you absolutely need to write C code wrap it in a class and implement a thought-out interface to that class. There is really no reason to write C code at all except in the case of `getopt_long(3)` (see below). Also *do not* use C-style casts in your code. If you need to cast use `dynamic_cast` or, failing that, `static_cast`.

Remember: this is a course focusing on Linux software development using C++.

The particulars

- You will find an `.ice` file in the exercise tarball.
- Figure out how to use Ice and how to use the interface.
- We have set up a provider node for testing:
  - The endpoint is `Provider:ssl -h lui.inso.tuwien.ac.at -p 3434`.
  - To create accounts on the provider node, visit `https://lui.inso.tuwien.ac.at/`.
  - You *must* connect via SSL and you *must* use a client certificate downloaded from the above URL. The IceSSL plugin is used for transport layer security.
- Configure and initialise the Ice runtime properly and set up the needed plugins (IceSSL). The Ice manual has a detailed chapter explaining all you need to know about properties and configuration options. If you use commandline options to set Ice properties be sure to initialise the runtime *before* passing any arguments to your commandline handler (see next item) so they can be stripped from `argv` by Ice:

  > [...] This means that you should initialize the Ice run time before you parse the command line for your application-specific arguments. That way, the Ice-related options are stripped from the argument vector for you so you do not need to explicitly skip them. [...]

- Use `getopt_long(3)` for argument handling:
  - The object identifier string used to connect to the EHR provider node must be an argument.

    The long option has to be `--node`.
  - If the option `--list` is given, display a list of all documents available in the patient's EHR on standard output.
  - The ID of the document selected for exporting to PDF is given using the long option `--document`.
  - The output file is given using the long option `--output`.
  - The options `--list` and `--document` are mutually exclusive and `--document` needs `--output` (and vice-versa).
  - The PEM files containing the requestors and the patients certificate and private key for signing are given using the long options `--requestor` and `--owner` respectively.
  - If the long option `--help` is given display a usage message on standard error (`stderr`, `cerr`, file descriptor 2).

    Pass your favourite UNIX utility the option `--help` and see what it displays to have an idea what the usage message should look like.
  - Wrap the `getopt_long(3)` functionality in a C++ class and use getters to access the values supplied by the user.

  - An example invocation of the program would be:

    ```
    ehr_client --node "Provider:ssl -h lui -p 3434" \
    --requestor pharm.pem --owner pat.pem \
    --document 7 --output foo.pdf
    ```

    The backslashes denote that all of the above should be on one single line.

- Use Cairomm to generate a nice PDF.
  - Isolate rendering of the document into a C++ class.
  - When designing the class, keep in mind that Cairo supports various formats, and that you should be able to render to different surfaces in the upcoming tasks.
  - You can find Cairomm example code in the exercise tarball.

- Use OpenSSL for cryptographic functions. We have created a C++ wrapper that you are supposed to use in your code. The wrapper class is named `Security`, and can be found it in the exercise tarball. Note that this has nothing to do with IceSSL and transport layer security. The cryptographic functions provided in the `Security` class are used to encrypt and decrypt documents in the EHR system and generate and verify digital signatures for requests.

- Use classes when designing your program. You will find a lot of example code on the internet that is procedural; do not reuse it directly. Avoid mixing C and C++ code. Wrap C code into classes or functions if needed.

- Handle exceptions properly

- Handle errors correctly
  - Your program should not crash
  - Your program should correctly deal with missing mandatory options, unknown options, invalid files, ... An important aspect you should keep in mind is that the user interface of your program is defined by the command line processing you implemented. Be sure to invest some time to test if the behaviour is correct.

    Test all possible combinations of command line arguments. For example make sure that a user can't supply the same command line option twice or more without getting a usage error.

1.4.3 Using version control

It is important to actually *use* version control. Committing patch sets implementing one feature or fixing a specific bug is good version control usage. Do not commit one huge chunk of code hours before the deadline. Please attend the Subversion lecture for more information.

- Read the relevant parts of the Subversion book [7].
- Initially populate the empty repository.
- Use a `trunk/` directory. It makes life easier.
- The first rule for commit messages is that they *must not be empty*. Please write meaningful messages.
- Another rule is not to commit any binary files that can be built from source or by other means from files that are checked into your repository. Ideally your repository contains only text-based files such as source and header files, and Makefiles describing how to build binaries from those sources.

Please note: Do *not* commit your private key to the repository. Ideally you wouldn't even store your key in your home directory on the shell servers but that would be too inconvenient for working and developing. We have, however, made sure that no tutors or other students have access to your home directory, thus making your private keys relatively safe. The repository, however, is accessible by all tutors since we use it to look at your code and evaluate and rate your submissions. Thus it would be very bad security practice to commit data as sensitive and secret as your private key.

So please, *do not* commit your private key. Keep it as safe and to yourself as possible.

1.4.4 Building your programs

The build system is a very important aspect of the first task. The best software is worth nothing if no one can build it.

- Read the GNU make info pages (`info make`)
- Create a build system for your sources
  - You have to manually create a Makefile building your client. Manually means that no tools may be used for creation of the file. Call your Makefile `Makefile.man`
  - Keep an eye on the dependencies of the targets.
  - Use the `-Wall` flag when compiling with `g++`. It enables all warnings about constructs that some users consider questionable. Fix the warnings.
  - Your source must build at least on the Linux and BSD shell servers by just typing `make -f Makefile.man` or `gmake -f Makefile.man`.
  - It is intention that some libraries are installed in non-standard paths on the shell servers. You will need to set the correct compiler flags (paths) in your Makefile for it to work on both platforms. You will probably also need to set the environment variable `LD_LIBRARY_PATH`. Please attend the GNU Make lecture for more information.

1.5 Deliverables

Tag the source code in your subversion repository in `/submission/task1_final`. If you don't yet know what tagging means please consult the Subversion book [7, Ch. 4 *Tags*].

References

[1] Geotechnical Software Services: *C++ Programming Style Guidelines*. http://geosoft.no/development/cppstyle.html. Version: 2008. – [Online; accessed 2009-03-18]

[2] Henning, Michi ; Spruiell, Mark: *Distributed Programming with Ice*. http://www.zeroc.com/doc/Ice-3.3.0-IceTouch/manual/. Version: 2008. – [Online; accessed 2009-03-18]

[3] horstmann.com: *Moving from Java to C++*. http://www.horstmann.com/ccj2/ccjapp3.html. Version: 2007. – [Online; accessed 2007-02-26]

[4] Stroustrup, Bjarne: *The C++ Programming Language*. Special Third. Addison-Wesley, 1997 http://public.research.att.com/~bs/3rd.html

[5] Stroustrup, Bjarne: *The C++ Programming Language Online*. http://www.research.att.com/~bs/C++.html. Version: 2007. – [Online; accessed 2007-02-26]

[6] Sun Microsystems: *Code Conventions for the Java^{TM}Programming Language*. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html. Version: 1999. – [Online; accessed 2009-03-18]

[7] The Subversion Team: *Version Control with Subversion*. http://svnbook.red-bean.com/. Version: 2007. – [Online; accessed 2007-02-26]