Activity 2 - Basic Linux Networking

Daniel Schoonwinkel – E&E Postgraduate Networking Course 2019

May 28, 2018

1 Introduction

In this part of the networking course you will learn to install Linux programs, write, compile and execute very basic Python and C++ programs and use low-level networking sockets on Linux.

2 Practical examples

In this section we will present a walkthrough of commands and programs that will allow you to use Linux network sockets. The information contained in this section will enable you to complete the assignment at the end of this activity.

2.1 Installing programs in Linux

These are a couple of ways that you can install programs in Linux, and this section will discuss 3 ways: the XUbuntu package manager *APT*, compiling from source and installing using the Python packet manager *Pip*.

2.1.1 *Advanced Packaging Tool (APT)*

Most Linux distributions have a package manager. Package managers are tools that provide pre-compiled binary files (executable programs and software libraries) as packages for your specific version of the operating system (OS). The managers of the your flavour of Linux - in XUbuntu's case that is Canonical - may impose some constraint or quality control requirement on binary files that are included as packages in the official package repository. Developers can also maintain their own repository, but Canonical then provides little guarantee of the usability or security of the external repository.

In XUbuntu, the package manager can be invoked from the commandline using `apt-` commands. For example, if you want to search for Wireshark packages, you would use the following command:

```
apt-cache search wireshark
```

This will show all the packages that are related to Wireshark in some way. More details on Wireshark will be given later in this walkthrough. Note that there are both programs and libraries included in the list. Libraries are indicated with the 'lib' prefix.

To install a program using *APT*, you would use the following command:

```
sudo apt-get install wireshark
```

Note that the command `sudo` is added before the install command: this is necessary because only the root user or users with root privileges are allowed to install packages. On Linux, the root user is similar to an Admininstrator on Windows.

You should now be able to run Wireshark from the commandline as follows:

```
sudo wireshark
```

We will use Wireshark later, for close the program and continue with the walkthrough.

As you can see, using the package manager is very easy and no extra configuration is needed. The package manager is generally the preferred method of installing programs on Linux.

However, at times the program that you require is not available in the official XUbuntu repository. For example, *Sublime Text 3* is not available in the official repository, but in a separate developer repository.

The instructions for using this repository is at
https://www.sublimetext.com/docs/3/linux_repositories.html .

Go ahead and install *Sublime* now according to the `apt` instructions.

We suggest using *Sublime* for text editing and programming as it is lightweight and performs syntax highlighting in various computer programming languages. Note: it is not a complete IDE; compiling and running will be done from the commandline.

Finally, to remove a package, use:

```
sudo apt-get remove wireshark
```

2.1.2 *Pip* - Python's package manager

Pip is the Python equivalent of `apt`, but only for Python modules. *Pip* is cross-platform (as is Python), and can therefore be used on Windows and Mac OSX as well. It works with the following commands (*Pip* might need to be installed with `apt` before it is availabe for use):

```
pip3 list #lists the installed python packages
pip3 search pypacker #shows all packages related to pypacker
pip3 install pypacker #installs package
pip3 uninstall pypacker #removes package
```

See [1] for more details on `pip3`.

2.1.3 Compiling and Installing from source

If a program is not in an Xubuntu repository or you need to configure it in a special way, you will have to compile the binaries from the source code. We will install *Libtins* (a C++ packet crafting library) to illustrate this. The instructions can be found here: https://libtins.github.io/download/

You will frequently see installation instructions similar to this format:

1. **install dependencies** : install all the necessary dependencies through *APT* or compile them from source.

2. `configure` : this is an automatic script that will detect which operating system you use and which utilities / dependencies are available. If a critical dependency is missing, it will stop the build progress and notify the user which dependency could not be found. If everything is in order and ready for the build process, it usually creates a Makefile or some configuration file that the `make` step cannot run without.

3. `make` : This is the compiling step, and will probably take the longest. Usually multiple source files and folders are included in the build step. If compilation of one of the source files fail, the build is usually stopped with an error.

4. `make install` : the final step in the build process, installing the library and binaries in a suitable place. Usually this is in /usr/local/lib and /usr/local/bin or similar. Depending on the write permissions of that folder, you will have to sometimes run `sudo make install`.

Not all of these steps are always included and extra steps could also be included as with `cmake` in the *Libtins* installation.

2.2 Writing and running Python and C++ programs on Linux

Now that you can install programs on Linux and are comfortable with the commandline, let us discuss writing programs using *Sublime Text* and the commandline.

2.2.1 Python Hello World!

Python is high-level language that is easy to learn and understand. It emphasises code readability and therefore removes some of the typical punctuation and syntax of other more formal programming languages. See [2] for more information.

Writing programs in Python is very very simple:

```
# Run:
# python3 helloworld.py

## Import other modules and programs here

# The main function creates a logical starting point (for debugging etc.)
def main():
    print("Hello World!")


#This function will only be called if it is run as a program,
# NOT when it is included as a library / module in other code
if __name__ == '__main__':
    main()
```

The program above is also included in the activity directory. Be aware that all Python programs can be included in other programs by using the `import`

command. For example, test what happens when you run `helloworld_import.py` All lines at global scope are run even if it is just imported.

Therefore the use of the "If MAIN" clause: it prevents the `main` function to be run if it is not used as a standalone program.

Python is generally easier to get started with, and easier to debug. It is recommended as the language to prototype your ideas or for simple projects. It has lots of functionality and also be used for more complex projects if they are not too computationally expensive.

The main disadvantages of Python are its longer latency times in networking and in some cases slower than C++ computation. Python does support offloading heavy computations to C++ modules with very little extra performance penalties.

In terms of programming complexity, the following options can be considered, from easiest to most difficult:

- Write all of your code in Python if performance is not an issue. Always try this option first and then decide if performance is indeed an issue. Use profiling modules like `profile`, `cProfile` and texttttimeit to measure which part of your code takes the longest. See https://docs.python.org/3/library/profile.html for more details on profilers.
- Write the slowest parts of your code in Python compatible C++ code. Test if this solves your problem.
- If performance is a major issue, rewrite the program is C++. Even in this case the Python program can help: getting an idea of how the functions and variables are structured in Python will help with writing the C++ program.

2.2.2 C++ Hello World!

C++ is a powerful general-purpose language. It is an extension of the C language, to include object-orientated features. Its latest standard is C++17, but we will be using C++11 standard in this course because it is already supported by most C++ compilers. See [3] for more details.

C++ can manipulate network packets on the raw Ethernet packet layer or higher (IP, UDP, TCP) layers using OS network sockets or use various C++ libraries such as *Boost*[4] and *Libtins*[5].

For now we will only write our first C++ program:

```
//Compile with: g++ -o helloworld helloworld.cpp -std=c++11
//Run with: ./helloworld

//Library required for writing to stdout
#include <iostream>

int main(int argc, char *argv[])
{
        //Stdout stream, << streaming operator, and a newline character
        std::cout << "Hello World!" << std::endl;
```

```
        /* Show that the program was successful.
        This value can be read from commandline */
        return 123;
}
```

This file is also in the activity directory. Note the instructions on how to compile and run the program at the top of the file. Run the given commands on the commandline and in the same directory as the helloworld.cpp file.

Unlike the Python program, this program needs to be compiled before it can be run. This is where some of the efficiency of C++ lies: the compiler performs strict syntax checking to ensure that the instructions in the program is executed accurately. Depending on the compiler, the code might be optimised before it is compiled into binary code. These binaries are built for the computer's specific hardware configuration, so further efficiency can be unlocked. A program compiled on one architecture can usually not be run on a different computer.

2.2.3 Compiling and GNU Make

3 Assigment 2: Write your own ping-request and ping-responder.

4 References

List of sources:

[1]https://pip.pypa.io/en/stable/user_guide/

[2]https://www.python.org/about/

[3]http://www.cplusplus.com/info/description/

[4]https://www.boost.org/

[5]https://libtins.github.io/