Hardware and Software Requirements List

For the book "Hands-On System Programming with Linux", Kaiwan N Billimoria, Packt (2018).

The complete source code for this book - *Hands-On System Programming with Linux, Kaiwan N Billimoria, Packt*, is available on it's GitHub repository. You can download and work on it by cloning the git tree like so:

```
git clone
 https://github.com/PacktPublishing/Hands-on-System-Programming-with-Linux
```

(type the above in one line please).

For the best experience in trying out the provided sample source code in this book (and writing your own along similar lines), we recommend the following hardware and software. Following that, we list a few 'required' and 'optional' software packages (or utilities).

Hardware

You will require a modern desktop PC or laptop; *Ubuntu Desktop* specifies the following as *"Recommended system requirements"* for installation and usage of the distribution:

- 2 GHz dual core processor or better
- RAM
    - running on physical host: 2 GB or more system memory
    - running as a guest **VM (Virtual Machine)**: the host system should have at least 4 GB RAM (the more, the better and smoother the experience)
- 25 GB of free hard drive space
- Either a DVD drive or a USB port for the installer media
- Internet access is definitely helpful.

Running Linux as a Guest VM

A practical, convenient alternative to using a native Linux system: one can install and use the Linux distribution as a guest OS on a virtual machine (VM or guest). We recommend using *Oracle VirtualBox 5.x* (or whichever is the latest stable version available). The host system should be either MS Windows 7 or later (practically, Win 7 Pro or Win 10 is preferred), or a recent Linux distribution.

Oracle VirtualBox download link: https://www.virtualbox.org/wiki/Downloads

Again, *Oracle VirtualBox* is considered OSS (Open Source Software) and is licensed under the GPLv2 (same as the Linux kernel); it too is free.

VirtualBox Documentation: https://www.virtualbox.org/wiki/End-user_documentation

Software

We recommend the reader use one of the following Linux distributions (can always be installed as a guest OS on a Windows or Linux host system, as mentioned above):

- Ubuntu 18.04 LTS Desktop
- The Ubuntu 16.04 LTS Desktop is a good choice too (it has LTS - Long Term Support - as well), and everything should work

Ubuntu Desktop download link: https://www.ubuntu.com/download/desktop

- Fedora 27 or 28 (Workstation)

Download link: https://getfedora.org/en_GB/workstation/download/

Note:

- These distributions are, in their default form, OSS and non-proprietary, and free to use as an end-user
- Though our aim is to be Linux 'distro' neutral, the code has only been (lightly) tested on Ubuntu 18.04 LTS and Fedora 27/28
- Most of the code output we have shown in the book is off an Ubuntu Linux guest (due to Ubuntu's popularity); the reader should realize that due to (usually minor) distro - and even within the sam distro, differing versions - differences, the output may not perfectly match what you see on your Linux system.

Software Packages

There are two categories: *required* and *optional*.

Required Packages

The packages that get installed by default when one uses the typical Linux desktop distribution (like any recent Ubuntu or Fedora Linux) will include the minimal set required by a systems programmer: the native *toolchain*, which includes the `gcc` compiler (along with headers) along with the `make` utility. On occasion (for example, in *Ch 4*), we build a 32-bit binary on a 64-bit (x86_64) system. To do so successfully requires the appropriate toolchain support.

- On Ubuntu Linux, please install the required package like this:

  ```
  sudo apt install gcc-multilib
  ```

In *Ch 8, Process Capabilities,* we show an example of file capability bits embedded in a binary executable called `dumpcap(1)`. Seeing this for yourself requires `dumpcap(1)` installed, which in turn requires the *wireshark* package to be installed and configured correctly.

- Install the wireshark package on Ubuntu with:

  ```
  sudo apt install wireshark-common
  ```

  During installation, in response to a question on whether you want *wireshark* to capture packets as non-superuser, select "Yes". Else (if perhaps already installed), you can reconfigure wireshark like so:

  ```
  sudo dpkg-reconfigure wireshark-common
  ```

  selecting "Yes" to this question. (Only now will the `dumpcap(1)` binary will get embedded with the capability bits we expect and demonstrate in the chapter).

In *Ch 9, Process Execution*

- We require the `getcap(1)` and `setcap(1)` utilities; install them on Ubuntu with:

  ```
  sudo apt install libcap2-bin
  ```

- One of the exercises (again in Ch 9) requires the *Poppler* package (PDF utils) to be installed; it can be installed as follows:
  - On Ubuntu: `sudo apt install poppler-utils`
  - On Fedora: `sudo dnf install poppler-utils-<version#>`

  A tip: above, for the Fedora case: for getting the version number, just type the above command and after typing `poppler-utils-` press the `[Tab]` key twice; it will auto-complete providing a list of choices; choose the latest version and press `[Enter]`.

- **Clang compiler frontend**

LLVM/Clang is an open source compiler for 'C'. We do use the clang compiler, notably in *Chapter 5, Debugging Tools for Memory Issues* especially for using the "sanitizer" compiler- instrumentation toolset. It remains useful throughout the book (and indeed is used in many of our `Makefiles`), thus installing clang on your Linux development system would be a good idea!

It is not completely essential and one can stick with familiar `gcc` too - provided one is willing to edit the Makefile(s) to switch back to gcc wherever required! Installing `clang` on the Ubuntu 18.04 LTS desktop is easy:

```
sudo apt install clang
```

Clang documentation can be found here: https://clang.llvm.org/docs/index.html .

- **llvm-symbolizer**: this tool requires to be installed as well (when used appropriately, it provides useful debug information of the form `filename:line#`). Install it on Ubuntu Linux with:

  ```
  sudo apt install llvm-symbolizer
  ```

Optional Packages

The book, at times, mentions that running a program on another CPU architecture - typically ARM - might be a useful exercise; it's not considered mandatory of course. If you wish to try (interesting!) stuff like this, please read on, else just skip this section.

One way to try things on an ARM machine is to actually do so, which implies you have a physical ARM-based single board computer (as an example, the *Raspberry Pi* is very popular).

Qemu

Often, an easier way to just try things out, is to have an ARM/Linux system *emulated* - this alleviates the need for hardware! To do so, we recommend using the superb *Qemu* project (https://www.qemu.org/) .

It's much easier though, to just install the required *qemu* packages:

On Ubuntu: `sudo apt-get install qemu-system-arm`

Fedora: `sudo dnf install qemu-system-arm-<version#>` (same *Tip* as previously mentioned applies).

Cross Compiler

If you intend to write a 'C' program that is compiled on an x86_64 system but runs on something else (say, an ARM-32), then you require a *cross compiler / toolchain* installed.

On Ubuntu, install it with:

```
sudo apt install gcc-arm-linux-gnueabi binutils-arm-linux-gnueabi
```

On Fedora, install it with:

```
sudo dnf install arm-none-eabi-binutils-cs-1\:2.28-3.fc27.x86_64
```

```
  arm-none-eabi-gcc-cs-1\:7.1.0-5.fc27.x86_64
```

The above example was for Fedora 27 particularly; in general, the syntax is:

```
sudo dnf install arm-none-eabi-binutils-cs-<version#>
  arm-none- eabi-gcc-cs-<version-#>
```

Tip: Above, for the Fedora case: for getting the version number, just type the above command and after typing `arm-none-eabi-binutils-` press the `[Tab]` key twice; it will auto-complete providing a list of choices; choose the latest version and press `[Enter]`. (Same goes for the `gcc` package).

Toolchain Resources

In general, a good resource on toolchains: https://elinux.org/Toolchains.

A 'wiki' page (from the *SEALS* project I maintain) with instructions on downloading and installing the latest ARM Linaro toolchain is available here:

https://github.com/kaiwan/seals/wiki/HOWTO-Install-required-packages-on-the-Host-for-SEALS.

Miscellaneous

Also mentioned at times in the book, your author has created and maintains a small project on GitHub - ***SEALS*** *(Simple Embedded ARM Linux System*): https://github.com/kaiwan/seals/.

SEALS is a very simple, "skeleton" Linux. It uses a bash script that internally cross-compiles a Linux kernel for ARM, creates and initializes a simple root filesystem, and then calls upon Qemu to emulate and run an ARM platform (the Versatile Express CA-9); the useful thing is, the script builds the target kernel, root filesystem, the root filesystem image file, and sets things up for boot. It even has a simple GUI (or console) front-end, to make configuration a bit simpler for the end-user. Clone it and give it a try... we defintely recommend you have a look at its *Wiki* section pages for help.

Ubuntu: the package **libncurses5-dev** is required is you intend to configure the kernel via `make menuconfig`; we do not require to do so by default. If required:

```
sudo apt install libncurses5-dev
```

In a few places in the book, we try and illustrate the true situation better by making use of some pretty useful and powerful tools; one of them is `perf(1)` for (CPU) profiling. Installing **perf** on Ubuntu is simple:

```
sudo apt install linux-tools-$(uname -r)
```

Another superb tool for tracing is the powerful **LTTng** (Linux tracing Toolkit next generation) toolset. In order to learn how to install and use it, we refer you to its (very good) documentation:

https://lttng.org/docs/

Ubuntu: the *System Monitor GUI* application is first used in *Ch 14, Multithreading Part I - Essentials* to demonstrate CPU usage for a matrix multiplication done with and without threads (single vs multithreaded). If not already installed, you can install it with:

```
sudo apt install gnome-system-monitor
```

Ch 17, CPU Scheduling briefly demonstrates usage of the `chrt(1)` and `taskset(1)` utilities. Both the *chrt* utility (show and set process scheduling policy and priority) and the *taskset* utility (show and set process CPU affinity) can be installed by installing the `util-linux` package on both Ubuntu and Fedora.

The Source Code for this Book

The complete source code for this book - *"Hands-On System Programming with Linux", Kaiwan N Billimoria, Packt* - is available on GitHub; you can download and work on it by cloning the git tree like so:

```
git clone
 https://github.com/PacktPublishing/Hands-on-System-Programming-with-Linux
```

(type the above in one line please).

The source code is organized chapter-wise; each chapter is represented as a directory, f.e. `ch8/` has the source code for *Chapter 8*. The root of the source tree has some code that is common to all chapters - `common.c` and `common.h`.

For efficient code browsing, you could always index the codebase with *ctags* and/or *cscope*. For example, for ctags:

```
ctags -R
```

in the root of the source tree is sufficient.

To build the code, for say, *Ch 8* (the *git clone* is required only once):

```
git clone https://github.com/PacktPublishing/Hands-on-System-Programming-with-Linux
cd ch8
make
```

[...]

Enjoy.

[End doc]