Global Scientific JOURNALS

GSJ: Volume 5, Issue 9, September 2017, Online: ISSN 2320-9186
www.globalscientificjournal.com

DEVELOPMENT OF A GENERIC HIGH LEVEL APPLICATION PACKAGE INTERFACE ON LINUX-BASED OPERATING ENVIRONMENT

Gbadamosi, O.A., Ilori, A.O. and Ayegbusi, O.A.

KeyWords
Application Package Interface, Linux, Operating System Environment, libfprint, MINDTCT, BOZORTH3, C Programming Language.

ABSTRACT

Linux-based operating environment in devices is becoming more popular especially on standard desktop computers and laptops. These Linux operating system are free distributabable and can be installed even on mobile and tablet devices. However, despite the uniqueness of these operating system software, its support on hardware components such as fingerprint scanners seems stringent. Fingerprint technology has become more useful even in many house-hold hard-ware devices and also in many technologies used in various sectors of the economy ranging from education, banking, security and so on. This study is aimed at developing and implementing a generic high level application package inter-face to make commonly used hardware devices like finger print scanner usable on Linux-based operating environment. The results show that this study work can serves as basis to implement other hardware components usable on many other operating environments.

1.0 Introduction

Linux operating system is a freely distributable, cross-platform operating system based on Unix that can be installed on PCs, laptops, netbooks, mobile and tablet devices, video game consoles, servers, supercomputers and more. The popularity of Linux on standard desktop computers and laptops has been increasing over the years [Hoffman and Mitran, 2015]. Many applications available for Microsoft Windows and Mac OS X also run on Linux. However, some commonly used hardware like fingerprint scanners does not run on Linux-based OS.

In many part of the world, there is any sector that is yet to fully implement the use of fingerprint devices either as security measure or to curb impersonation ranging from the education, military, banking sectors and so on. Fingerprints are distinctive patterns that are fully developed from birth and are permanent throughout the life span. These distinctive patterns consist of ridges and valleys. Fingerprints are unique for individuals since no two persons have the same [Pavel, 2006].

These uniqueness of fingerprints makes it more reliable than many other protective measures such as passwords, id cards etc. However, it can as well be used to augment an existing authentication system as an additional level security [Daniel, 2007]. An application program interface (API) is a set of routines, protocols and tools for building software applications. API specifies how software components interacts. Just as a Graphical User Interface, an API makes it easier to use programs to build applications by abstracting the underlying implementation and only exposing objects that the developers need [West and Dedrick, 2001].

Despite the uniqueness of Linux-based software, it supports on hardware components such as fingerprint scanner seems stringent, it does not support many hardware devices. Fingerprints have replaced other methods of establishing the identities of human and the interest of taking unsupported devices and making them usable in Linux-based operating environment justifies this study. Other visible human characteristics such as facial features tend to change with age but fingerprints are relatively persistent [Jeff, 2008]. The API developed served as an interface between the fingerprint scanner and the Linux operating system, which will be inform of a driver. The driver interfaces are unified to a generic fingerprint scanning interface, and the interface is exposed to applications through a high-level API.

2.0 Materials and Method

The development of the generic application program interface in this study expresses the software components in terms of its operations, inputs, outputs and underlying types. This is done by separating codes specific to each supported device type into driver modules. UPEK TouchStrip (a simple type USB fingerprint scanner, which includes a biometric coprocessor which performs image processing in hardware) was used. The primitive operations of the design are shown in figure 1:

Fig 1: Primitive operations of UPEK TouchStrip

a. **Enroll finger: The fp_enroll_finger () function takes a device, performs enrollment, and upon success, returns some enroll**ment data in the form of a fp_print_data structure.

b. Verify finger: The fp_verify_finger () function takes a device and some enrollment data, performs a fingerprint scan, and returns a status code indicating whether the scanned finger matches the enrollment data.

c. Identify finger: The fp_identify_finger () function takes a device and a gallery (an array) of enrolled prints, performs a fingerprint scan, and returns the offset into the gallery where a matching print was found.

The TouchStrip driver will plug in as a `primitive' driver, but the imaging drivers will instead plug into an imaging layer that will convert them to fit the primitive driver interface, this act as a high level functionality for enrolling and verifying. The fingerprint scanner used was a USB device, hence there was no need for further abstraction for accessing devices over different buses. All USB devices identify themselves and their capabilities by presenting a device descriptor to the host system. The device descriptor includes a vendor ID and a product ID. MINDTCT and BOZORTH3 were used in the software development. MINDTCT is a minutiae detector which automatically locates and records ridge ending and bifurcations in a fingerprint image. BOZORTH3 is a fingerprint matching system which takes minutiae sets generated by MINDTCT and compares them for similarities. These two allows comparison to check if two fingers scanned are the same or not [Chen, Dass and Jain, 2005].

The generic high level API implemented as a library, compiled into a Dynamic Shared Object using C language. Fingerprint scanner applications to be drive will link against the library at compile time, and the run-time linker will complete the dynamic linking once the program is executed. The library is named `libfprint.'

libfprint depends upon two additional libraries at both compile time and run time:

- **libusb**, a shared library which allows performance of USB device I/O from userspace. This avoids having to write kernel-mode drivers for the fingerprint readers, and additionally paves the road for cross-platform compatibility.
- **glib**, a shared library providing miscellaneous utility functions. glib conveniently have quick access to linked lists, specialized memory allocators, string handling functions.

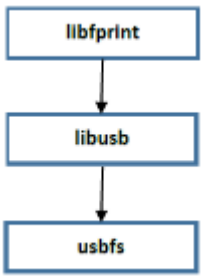

Fig 2: USB access layers

libfprint was implemented as a device-centric model. libfprint offers a fp_discover_devices () function call which scans the system for supported fingerprint scanners and returns them in a list of fp_dscv_dev structures. libfprint is able to produce a list of supported devices by checking USB device ID's. The fp_print_data_save () function which saves some enrollment data to the user's home directory, and the fp_print_data_load () function retrieves some previously saved data. Enroll data saved with the simple interface can be located with the fp_discover_prints () function, and then opened with fp_print_data_from_dscv_print ().

The imaging layer converts each fp_img_driver to an fp_driver structure and presents it to libfprint. The imaging layer provides generic implementations for the enroll/verify/identify operations which are expressed by a series of calls to driver-supplied fp_img_driver functions.

3.0 Results and Discussions

The imaging performance after implementation of the libfprint in the library function of the linux-based operating environment was excellent. The returned images are large and MINDTCT finds a large number of minutiae, leading to suitably distinct BOZORTH3 scores for matching/non-matching fingers. The outputs were shown in figure 3-6:

Figure 3: data enrollment page

Figure 4: enrollment output

Figure 5: verification page

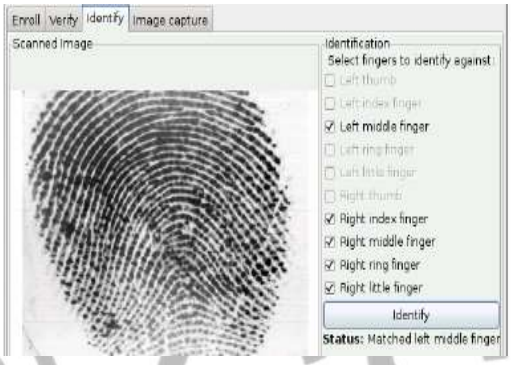

Figure 6: identification page

4.0 Conclusion and Recommendation

This work has helped to develop a generic high level application program interface for fingerprints on Linux-based operating environment. In the implementation stage using the library functions of the operating environment, no functionality was lost and the library is now suitable for integration into a wider range of applications. This work also serves as an example of how to integrate libfprint into more complex applications which can be implemented into other types of operating environment.

5.0. References

[1] Chen, Y., Dass, S. and Jain, A. (2005). Fingerprint Quality Indices for Predicting Authentication Performance, Proceedings of 5th International Conference of Audio and Video-Based Biometric Person Authentication, Rye Brook, NY. 160-170.

[2] Daniel Drake (2007). Usernameless Authentication. http://lists.reactivated.net/pipermail/fprint

[3] Hoffman Dale and Mitran Marcel (2015). Open Source and ISV Ecosystem Enablement for LinuxONE and IBM. Linux Foundation.

[4] Pavel Machek (2006). Driver for Thinkpad Fingerprint Sensor, http://lkml.org/lkml, Linux Kernel.

[5] Jeff White (2008). fprint for Embedded, http://lists.reactivated.net/pipermail/fprint.

[6] West Joel and Dedrick Jason (2001). Open Source Standardization: The Rise of Linux in the Network Era. Knowledge, Technology and Policy. Springer 14(2):88-112.