International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 6, June - 2013

Device Driver Wrapper

Sweety Raikesh
Dr. C.V. Raman University, Bilaspur (C.G.)

Abstract

A Device Driver Wrapper is software that functions as an adapter between an operating system and a driver, such as a device driver, that was not designed for that operating system. It can enable the operating system to use technologies for which no native implementation exists. Our project objective is to 'wrap' Windows based video device drivers so that they can be used as drivers in Linux. Wrapper works by emulating the Windows kernel and APIs, and dynamically linking the driver to this implementation. It implements Windows kernel API and the hardware device specific API within Linux kernel. A Windows driver for the device is then linked to this implementation so that the driver runs natively, as though it is in Windows, without binary emulation.

1. Introduction

In the current scenario hardware manufacturers provide drivers for Windows based systems while neglecting UNIX based systems e.g. Linux. As a result the current driver support for certain hardware devices in Linux is poor. Instead of rewriting the drivers for UNIX based systems, this project will aim to create a 'device wrapper' so that these Windows based drivers can be used on UNIX based systems.

This project is the ideal Linux solution to support devices for which no adequate native open-source drivers are available. It also allows vendors to drastically reduce time to market or eliminate the need to support multiple drivers for Windows and Linux. By using the same driver on both platforms, significant resources can be saved.

Thus it shall work as an emulation layer of the Windows based specification on a Linux based system, enabling Windows based drivers to work seamlessly in a Linux environment.

The Architecture of our system is shown in fig.1. In this paper, we will focus on the instance of wrapper for video devices supported by the AVStream specification in Windows. While the extension to other device architectures is left for further extension of the wrapper.

This paper is organized as follows. Section 2 overviews the wrapper architecture, and this section introduces the idea of wrapper and the structure of the system with wrapper introduced for Linux through the Video for Linux (V4L2) specification. In section 3, basic data structures and behaviour of each module are described. Related works are presented in section 4.

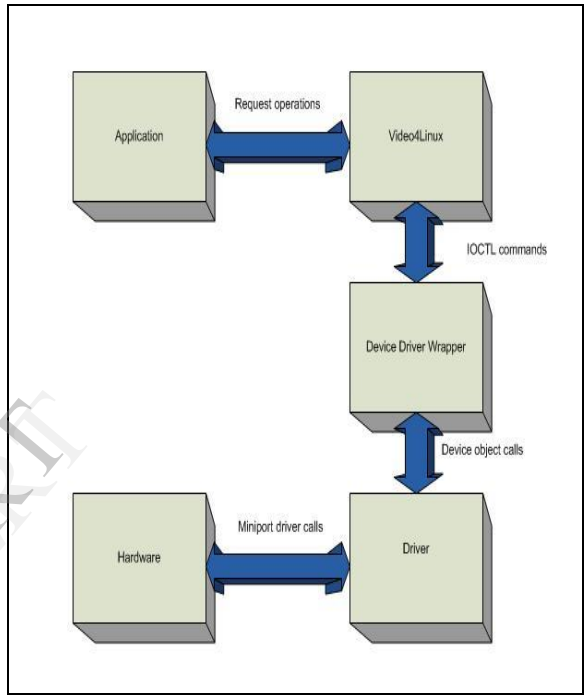

Fig.1.System architecture diagram

2. Literature Survey

A number of similarities exist between the Windows and Linux operating systems. On both systems, drivers are modular components that extend the functionality of the kernel. Communication between driver layers in Windows is through the use of I/O Request Packets (IRPs) supplied as arguments to standard system and driver defined functions, whereas in Linux function calls with parameters customized to a particular driver are used. Windows has separate kernel components that manage PnP, I/O and Power. These components send messages to drivers using IRPs at appropriate times.

In Linux, there is no clear distinction between layered modules, i.e. modules are not categorized as bus, functional or filter drivers. There is no clearly defined PnP or Power manager in the kernel that sends standardized messages to modules at appropriate times. The kernel may have modules loaded that implement Power Management or PnP

functionality, but the interface of these modules to drivers is not clearly specified.

Once data is passed to a driver that is part of a stack of modules by the kernel, the data may be shared with other drivers in the stack through an interface specific to that set of drivers. In both environments, hardware access through a HAL interface is implemented for the specific platform the kernel is compiled for, i.e. x86, SPARC etc. A common feature of both architectures is that drivers are modules that can be loaded into a kernel at runtime. Each module contains an entry point that the kernel knows to start code execution from. A module will also contain routines that the kernel knows to call when an I/O operation is requested to a device managed by that module. This enables the kernel to provide a device independent interface to the application layer. We exploit such interfaces to implement the Device Driver Wrapper.

3. Overview of Wrapper Architecture

4.

The various blocks of the architecture represented in fig.2 and the description of each is:

1) The wrapper initializer will initialize the wrapper with the linux kernel.
2) The loader loads the driver in the linux environment.
3) Linker module links the driver in the Portable Execuatble format in the Linux environment.
4) Interaction handler accepts the IOCTL commands from V4L, and is responsible to return the output to the given layer.
5) Windows kernel level simulator adjusts the generic Linux Windows compatibility such as register stack handling,shadow space adjustment.Also it provides 64-bit compatibility and 32-bit & 64-bit inter-arithmetic.
6) Emulations functions will emulate the AVStream calls taken as V4L calls from the I/O handler and provide them to miniport driver handler.
7) The miniport handler driver interacts with the miniport driver through packets.It also interacts with the HAL if required.

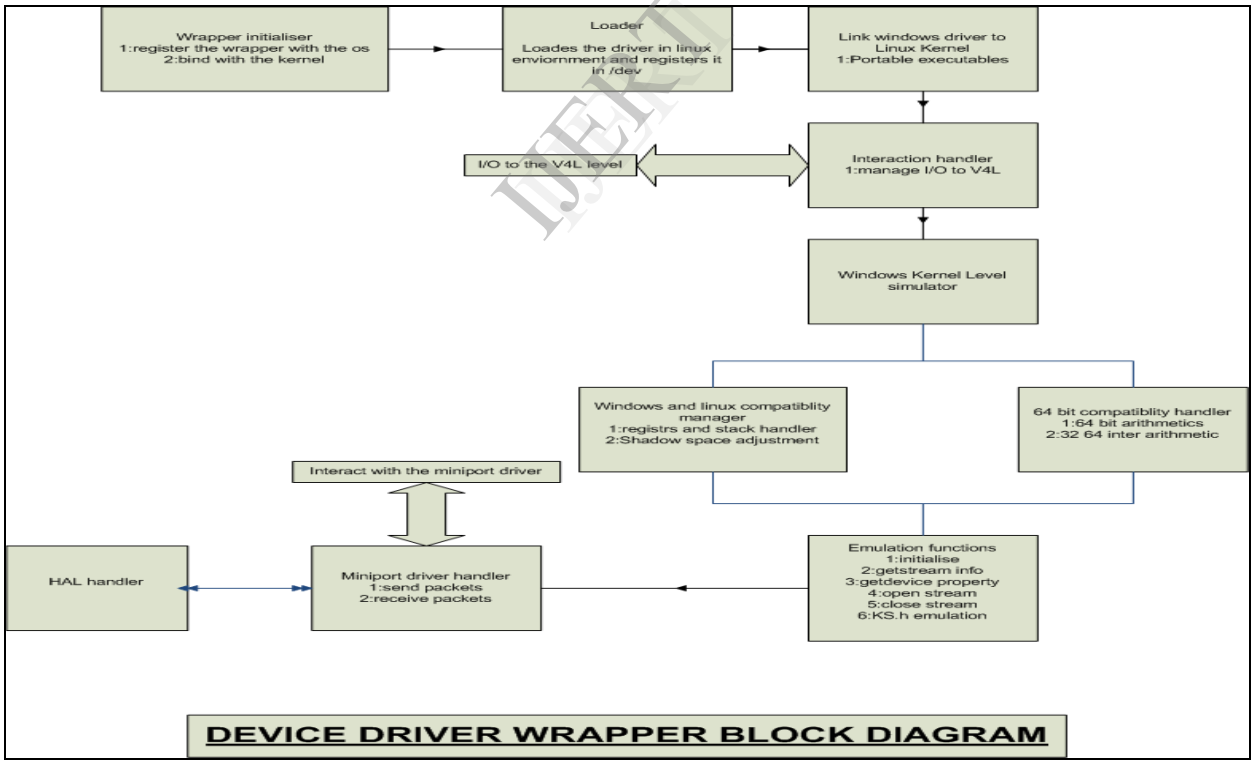

Fig.2. Wrapper Architecture diagram

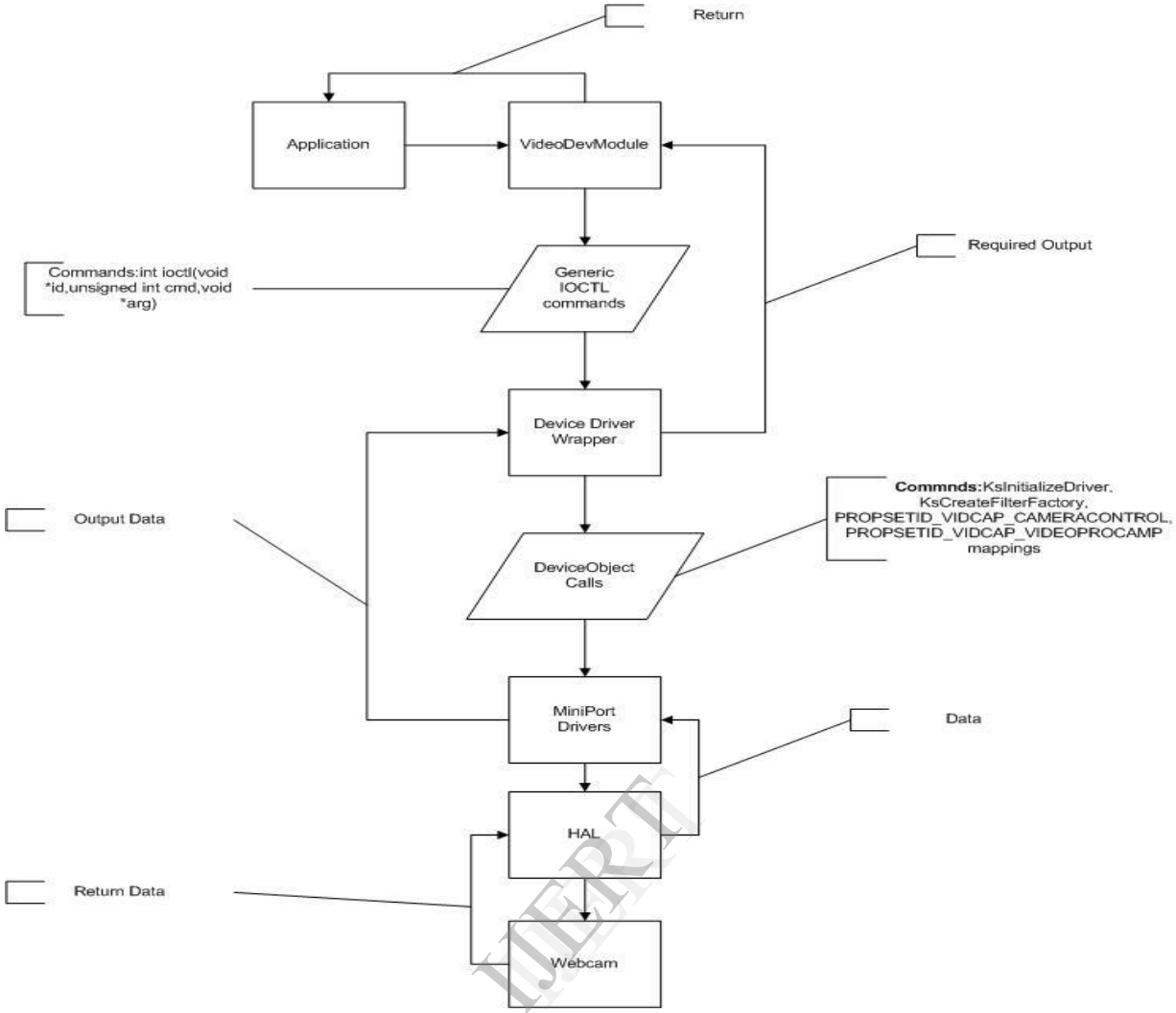

Fig. 3 Data Flow Diagram

5. The Data Flow

As shown in figure 3, the data flow diagram can be interpreted as follows:

1) The application provides data to the VideoDev module in V4L to send the requests specific to the device.
2) The VideoDev module then sends various IOCTL commands to the device wrapper.
3) The device wrapper creates a device object for the device and transforms these calls to Windows specific device object calls.
4) These Windows specific calls are accepted by the miniport driver to perform the requested operation. The miniport driver gives these corresponding calls to the hardware device through the HAL.
5) The output is provided from the HAL to the VideoDev module through the device wrapper and finally returned to the application.

6. Conclusion and Future Work

In this paper, we present the architecture of wrapper. The purpose of this was to build an application that would enable Linux users to use the hardware with Windows based drivers seamlessly. In addition, factors like generic Windows kernel simulation, 64 bit compatibility were key points to be taken into account. Studies have shown us that the wrapper can help save a lot of time and effort required in reverse engineering with the quality features provided by the specific vendors can be used to the fullest. Each video device having a windows based driver can be used with Linux thus enhancing the streaming media devices support in Linux.

The AVStream wrapper can be extended for other streaming media devices with certain extensions. Extending the support to other architectures and its feasibility study can be carried out. Future work can include the analysis and resolution of these problems.

7. Acknowledgement

I would like to express my immeasurable thanks to my internal project guide Prof. Mr. Praveen Chouksey for his guidance and excellent suggestions in the course of the development of my project. I would also like to thank my external guides Mr. Jitendra Asrani and Mr. Someshwar Jaju for providing me with the opportunity to work on this project and for their guidance. I am also very thankful to Prof. Mr. Tarun Dhar Diwan, Coordinator of the M.Tech (S.E.), for providing all the necessary facilities, conducting reviews and giving invaluable feedback on the project. Finally, I would like to thank my staff members, friends and all other people who have directly or indirectly helped me in the design, implementation and deployment of the project.

8. References

[1] *Linux Device Drivers, 3rd Edition By Jonathan Courbet, Greg Kroah-Hartman, Alessandro Rubini*

[2] *Developing Drivers with the Microsoft Windows Driver Foundation by Penny Orwick and Guy Smith Microsoft Press 2007*

[3] *The Linux® Kernel Primer: A Top-Down Approach for x86 and PowerPC Architectures By Claudia Salzberg Rodriguez, Gordon Fischer, Steven Smolski*

[4] *Essential Linux Device Drivers by Sreekrishnan Venketeshwaran*

[5] *http:\\www.ndis.sourceforge.net*

[6] *A comparison of windows and Linux Device driver Architectures by Melekam Tsegaye and Richard Foss*

[7] *The Webcam HOW TO by Howard Shan*