IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-6, Special Issue-3, April 2018 E-ISSN: 2347-2693

Enhancement of Software and Data Portability by Normalizing Variations in Hardware

Pon Rahul M[1*], Rishi Shree S[2]

[1*]Dept. of Information Technology, Loyola-ICAM College of Engineering and Technology, Chennai, India
[2] Dept. of Information Technology, Loyola-ICAM College of Engineering and Technology, Chennai, India

[*]*Corresponding Author: theflyingrahul@icloud.com, Tel.: +919487323890*

Available online at: www.ijcseonline.org

***Abstract*—** This paper explores the concept of software and user data portability by tackling device driver issues caused due to diversity in computer hardware. The objective of this paper is to implement a universal interface between the hardware and the software intended to minimize time and manpower. Drivers are required for software-hardware integration and different hardware manufactured by OEMs require device drivers designed specifically for the hardware component. Development of a new platform requires the software manufacturer to target a variety of hardware. Upgrading existing software requires the device drivers to be checked if they are compatible with the newer version of the software. Device drivers have to be written for each software platform separately. This results in a lot of time being consumed which directly affects the consumer such as delayed software and firmware updates, security patches, bug fixes etc. This interface addresses the driver development issues by providing a standard hardware platform for which the software can be developed.

***Keywords*—** software, hardware, portability, platform, interface, drivers, compatibility

I. INTRODUCTION

Drivers are required for software-hardware communication, designed specifically for the hardware component. Different OEMs have been producing hardware components under different product portfolios. These components may have differences in features and product quality, but the final product is the same.

A. Kernel

It is a computer program and the core of a computer operating system which acts as the bridge for communications between the software and the hardware. It is usually the first program being loaded on start-up.

B. Device-Driver

Device driver is a computer program that operates a particular type of hardware device by providing a software interface (as a kernel module) for the installed hardware. It is a layer of hardware abstraction too.

Present modular design of the kernel allows any driver to be loaded or unloaded from the kernel memory space based on the demand. The problem with the concept of drivers is technology used in different hardware components and their functioning might differ from the typical one and this requires device drivers to be developed specifically for the component. Same products from different OEMs require different drivers and drivers are platform dependent. A single hardware driver cannot run across all software platforms. Plus the driver has to be made compatible to the newer version of the software if the software is undergoing a major upgrade. This diversity in devices bring up these problems:

1. Unnecessary bundling of basic drivers of various products with the operating system's kernel as modules.
2. Incompatibility of old drivers with a new operating system update.
3. Incompatibility of a hardware component with a platform's kernel where the manufacturer doesn't provide any device-drivers.
4. Failure of OEMs to provide firmware updates for older devices which might render them unusable with new software.

These issues unnecessarily increase the software product's package size due to driver bundling. Lack of driver support on platforms unsupported by the OEMs renders most hardware components unusable with other platforms' kernels, making the hardware partially/fully incompatible with the platform.

To tackle this issue, we propose the Project Causeway, a universal interface between the hardware and the software intended to minimize time and manpower. This interface addresses driver development issues by providing a standard hardware platform for which the software can be developed. We believe this decreases the effort in software development while making software and user data highly portable and hardware independent.

II. SUMMARY OF THE CONCEPT

In view of the above disadvantages present in existing technology, this innovation provides a common platform for which software development can be targeted to. Hardware variations can be normalized by providing a standard

interface called the CAUSEWAY INTERCONNECT BRIDGE, hereafter CIB. It has two ends, namely the software (S/W) end and the hardware (H/W) end. The S/W end is the same across all devices and serves as a universal hardware platform. The Operating System's kernel can be supplied only with essential drivers to connect to the S/W end. OEMs can target any platform by connecting their device to the H/W end of the bridge. The bridge has virtual ports via memory mapping to connect the H/W end and the S/W end. Thereby a virtual connection is achieved between the kernel and the hardware and further I/O can be made.

III. DESIGN

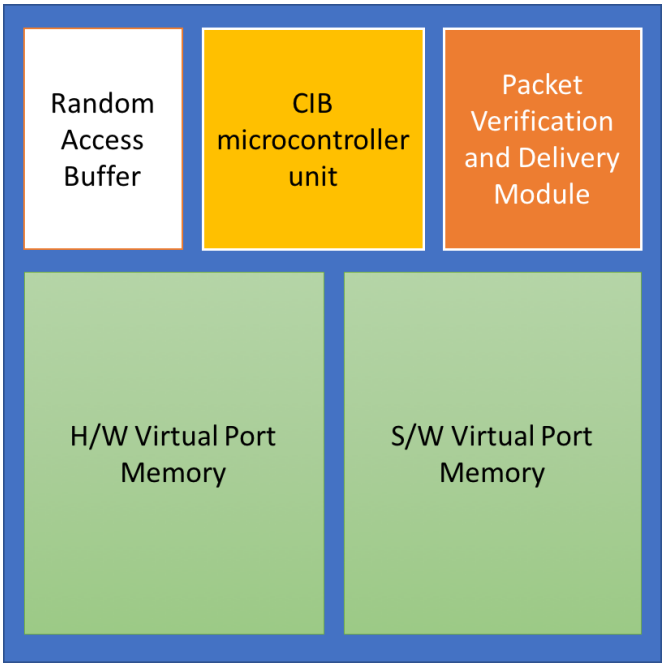

Figure 1. Chip design of Causeway Interconnect Bridge

The CIB will be a standalone hardware component of the mainboard of a computer with no control by the CPU. This can be achieved by using the PING PORT ALLOTMENT technique to allot ports for kernel-hardware connection. The chip will feature two virtual layers, completely abstracted from each other. These layers are RANDOM ACCESS MEMORY STREAMS. These memory streams will follow the QUEUE algorithm – First-In-First-Out (FIFO). First one is the software-end (S/W) and the other one is the hardware-end (H/W). The S/W end will act as a virtual hardware layer and as a standard hardware configuration for all software platforms. Software vendors will have to design the required STANDARD S/W-INTERFACE DRIVER for their kernel which will connect the kernel to the CIB. This driver will be the normalization layer for variations in hardware, thus making it universal. H/W end will be an OEM customizable layer which will connect their device to the CIB via the H/W-INTERFACE DRIVER supplied by them. These drivers will be stored in the devices' ROMs. The chip will come with a certain amount of superfast flash memory as buffer to help transfer instruction packet from the kernel to the hardware. The CIB chip will have a low-energy microcontroller to create new ports as per need and to clear existing ports and buffer memory once the I/O operation is complete. The chip will also have a PACKET VERIFICATION AND DELIVERY MODULE (PVDM) under the control of the CIB microcontroller to check the integrity of the packets and to ensure proper forwarding and delivery of the packets to and fro from the right hardware component. The CIB will have a control interface which is a part of BIOS/UEFI to facilitate firmware updates and patch deployment. Packet transmission from kernel to hardware and vice-versa has to follow a set of rules similar to the Transmission Control Protocol (TCP).

IV. WORK FLOW AND OPERATION

The overall operation workflow is similar to the functioning of a standard computer. A major change is done in the method of packet transfer from the kernel to the hardware and vice-versa using the CIB.

The steps by which this process is executed is stated below.

1. START
2. Application running above the kernel requests the kernel for connection to hardware for I/O (System Call).
3. Kernel loads the required hardware's STANDARD S/W-INTERFACE DRIVER into the KERNEL SPACE MEMORY.
4. Kernel forwards the request to the CPU for approval (Hardware Interrupt).
5. If the CPU approves the request, it forwards the interrupt to the requested hardware for new port creation in the CIB interface using "PING PORT ALLOTMENT" technique.
6. The hardware will connect to the CIB using H/W-INTERFACE DRIVER in its ROM.

Ping Port Allotment (PPA) algorithm

a. The CPU forwards the Hardware Interrupt directly to the requested hardware device.

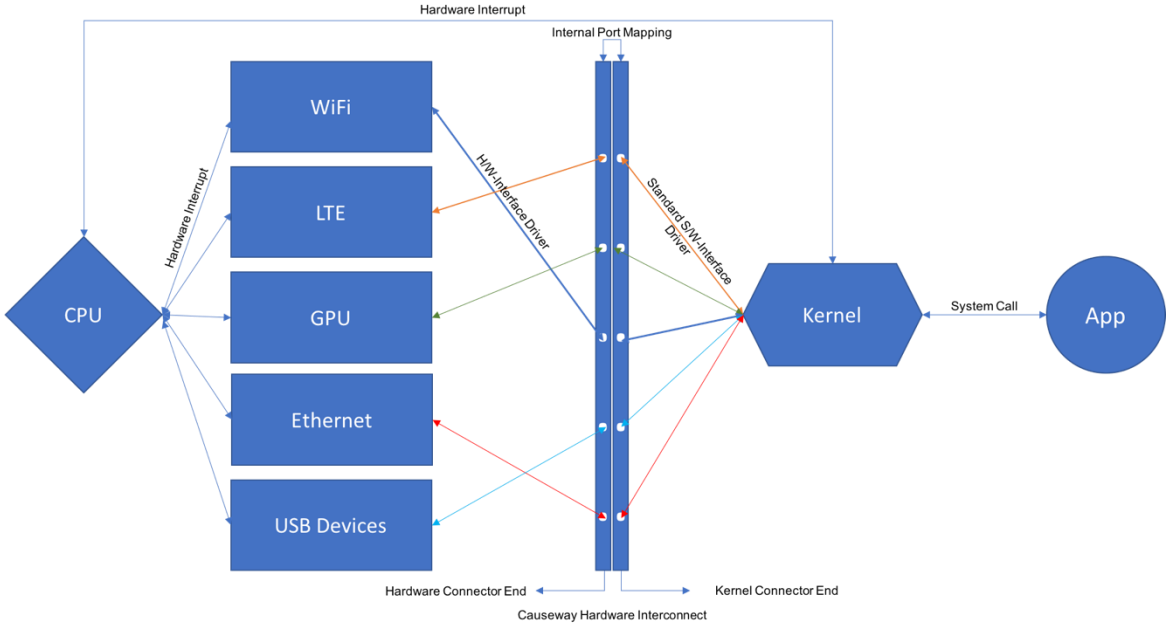

Figure 2. Flowchart representing workflow during the CIB interface operation

b. The device then attempts to connect to the CIB interface (H/W end).

c. If the CIB memory has sufficient free space for new virtual memory port creation, the microcontroller automatically assigns a new I/O port and returns it to the hardware component. NO TWO PROCESSES CAN HAPPEN THROUGH THE SAME PORT.

d. If the port memory is full, the request is stored in the buffer memory and awaits memory availability for some time. If memory is not available, the CIB microcontroller returns QUEUE FULL error.

7. Hardware device returns the virtual port information from the CIB microcontroller to the CPU.
8. CPU then forwards this port number to the software kernel.
9. Kernel now connects to the S/W end by loading the supplied Standard S/W Interface driver module.
10. Once the kernel and the hardware component are connected using the virtual port number, returned by the CIB microcontroller, the microcontroller takes the responsibility of data packet transfer between the kernel and the hardware device.
11. The microcontroller does a secure port mapping between the S/W and the H/W end and establishes a connection between them for the allotted port number.
12. The PVDM module, controlled by the CIB microcontroller, verifies the integrity of each packet and pushes it to the right port – to the right hardware from the kernel and vice versa.
13. If any packet is found missing, the kernel again requests the same packet from the device [1][2].
14. Once a I/O is completed successfully, the kernel compiles and verifies the integrity of the compiled data packets and sends a STOP packet to the CIB.
15. When the PVDM reads this STOP packet, it reports the same to the CIB microcontroller.
16. The microcontroller then severs the virtual connection between the two memory structures and pushes the port memory out of the queue, releasing free space for the next I/O request.
17. The device then gives a hardware interrupt signal to the CPU indicating I/O completion.
18. The CPU removes this completed process from the memory and clears all buffer files the from memory, if any.
19. This process is repeated again as per the need.
20. STOP

V. MODULAR DESIGN OF BRIDGE DRIVERS

Providing the kernel with a single driver to connect to the CIB will result in excess workload for the CIB microcontroller to

segregate and forward packets to the requested hardware. CIB is designed to be a low-power and an energy efficient chip. Therefore, this load can be minimized by segregating all information packets at the kernel and then sending them through the S/W layer. This is achieved by providing generic drivers for individual hardware components (say a printer or a network card) with the CIB interface as the target hardware specification. Drivers can be designed to be modular in nature. Therefore, the required generic driver can be loaded into the kernel as a module during the time of system call and can be unloaded from the memory once the I/O operation is complete [3]. This design also keeps the kernel space memory freer.

VI. VARIATIONS IN HARDWARE DESIGN

Present day's computer hardware designs are:

1. PC design:
    a. Devices soldered permanently to mainboard with no provision for customization.
    b. Devices connected to mainboard through the PCI or a similar slot with an option for expansion.
    c. Plug-n-Play hardware through USB or similar interface.
2. Mobile/System-on-Chip design.

Implementation of the CIB device architecture on these designs is almost the same from the hardware end. The variations are mentioned below:

A. PC design:

a. Permanently soldered hardware:

Since the hardware is permanently fixed, devices can be assigned with a permanent port number in the H/W end during the manufacturing process. Similarly all Standard S/W-Interface drivers can be assigned the same port number during the operating system installation. Therefore PPA is not required and only CPU's approval is enough for kernel-hardware connection.

b. PCI/ExpressCard or similar interface-based customizable hardware:

Hardware variations from user-to-user requires PPA logic to be followed to establish kernel-hardware connection. No port numbers can be pre-defined in this hardware design as the user may change his installed devices from time to time.

c. *Plug-n-Play hardware:*

This refers to removable hardware installed on the computer, like removable storage media or camera [4]. Plug-n-Play hardware will use the same logic as the previous case. Certain changes have to be done in the CIB's microcontroller firmware to provision dynamic hardware initialization and port allotment for such devices. This is because the CIB's microcontroller memory will keep a record on connected devices and Plug-n-Play devices need separate provision in the microcontroller firmware to dynamically include these devices in this record. And these devices have to be removed from the record once disconnected.

B. System-on-Chip design:

A typical SoC has all major hardware components like a LTE modem, a DSP (Digital Signal Processor), an ISP (Image Signal Processor), a SPU(Secure Processing Unit – Vault) and a TPM (Trusted Platform Module) bundled as a single unit along with the CPU. This is a more efficient design for light-weight, low-power handheld devices.[5] These chips can be targeted by permanently mapping the entire SoC to the H/W connector end of the CIB with the same logic used in permanently soldered hardware PC design. Permanent port numbers have to be allotted for different components of the SoC and necessary H/W-Interface drivers have to be provided.
These drivers can be upgraded to match the CIB microcontroller's firmware version through SoC firmware updates from the chip vendor.

VII. FIRMWARE UPDATE DELIVERY MECHANISM FOR CIB MICROCONTROLLER AND SYSTEM HARDWARE

Complete abstraction of hardware's vendor information and its specification demands a special logic for the delivery of firmware updates and patches to the hardware component's controller. Same logic can also be followed to update CIB microcontroller's firmware and apply patches for unforeseen vulnerabilities.
This can be made possible by assuming S/W updates are delivered on a timely basis. All firmware updates should be digitally signed [6] to expire on a specific date so that it will trigger the firmware update process. The entire update process will run below the kernel layer as a measure of hardware abstraction. This process must be integrated into the Power On Self-Test (POST) process so that all the firmware update processes can be performed before the operating system starts up.

A. Firmware Update and Patch Delivery (FUPD) Algorithm:

1. START
2. Hardware components' firmware signatures are verified during the POST process.

3. If the signature of the hardware component's firmware is expired, the device will request the CIB microcontroller for a update search.
4. The CIB will connect itself to the internet using the network hardware of the computer using the PPA logic and will search for the update package. This process will take place in the CIB's control interface embedded within the BIOS/UEFI as a measure of hardware abstraction.
5. If an update is found, the package is downloaded and forwarded to the target hardware component's reserve memory for unpacking and installing. This is managed by the PVDM module. This update might require a system reboot based on the hardware's role. A notification - in case of system reboot - will be forwarded to the kernel by the CIB microcontroller.
6. Enhanced POST is run again after the update to ensure proper functioning of all updated components before connecting the CIB's S/W end to the OS kernel. In case of any error, the performed update is rolled back.
7. If no updates are found, the microcontroller will self-sign the hardware's firmware for a grace period and it will continue checking for a new firmware update package and push them to the relevant hardware periodically.
8. Emergency patches are checked regularly on the internet by the microcontroller and deployed accordingly.
9. STOP

VIII. Comparison Between Legacy Hardware Model and CIB Model

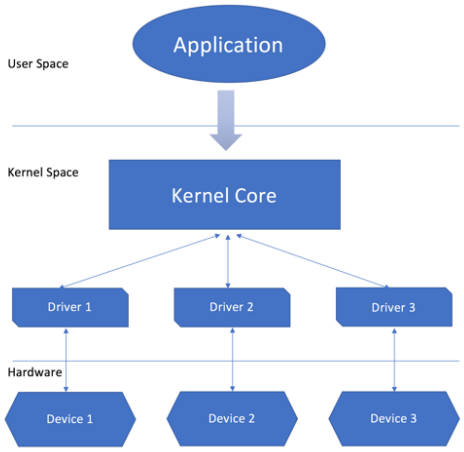

Figure 3. Flowchart representing workflow in legacy hardware model

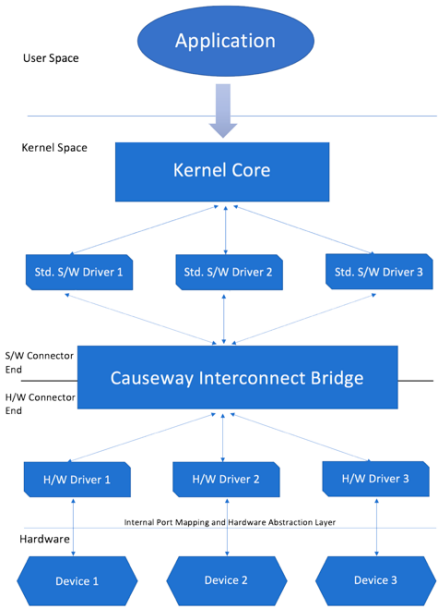

Figure 4. Flowchart representing workflow in CIB model

Tab. 1: Comparison between Legacy and CIB Design

Legacy Design	*CIB Design*
Kernel is connected directly to the hardware using device drivers.	Kernel is connected to the hardware using a dedicated chip which can handle I/O traffic.
Device drivers are specific in nature. Each hardware component may have a different design and may be manufactured by a different OEM, thereby requiring drivers built specifically for the hardware product. These drivers are also platform dependent.	It requires two set of drivers. One set is for the kernel-CIB chip connection which is universal for all devices belonging to the same class of hardware. These drivers are kernel dependent. Other set of drivers is for the CIB chip-hardware communication which are built by the manufacturers.
Hardware abstraction is not present.	Hardware abstraction is fully enforced.
Operating systems depend on the availability of compatible device drivers.	Operating systems have universal drivers bundled with the kernel, hence they don't have to compromise on driver availability.

IX. Significance of This Technology

As the title says, the very motto of this innovation is to enhance the user experience of the software by empowering every user to carry their own software in their pocket which can work across any hardware specification platform. Significances are mentioned below:

1. Cross-compatibility of a software product across any hardware specification with the same CPU architecture using the versatile CIB driver bundle in the kernel.
2. Software vendor can make their software work across all CPU architecture by providing multiple kernels which are compatible with most common architectures in the market (like AMD64 and ARM). This also enables faster software updates as developer can target all devices at once [7]. Software will become cheaper as the vendor no longer has to bundle any device-drivers with their product package.
3. Hardware information can be completely abstracted from the software using this technology as the software-kernel knows only the port number where the hardware is mapped to (only in case of permanently fixed hardware or SoC design).
4. The CIB interface will also act as an additional protection layer between the H/W and the S/W. Therefore, in case of any hardware compromise, I/O communications can be blocked at the CIB microcontroller, thus protecting data from being stolen.
5. This technology will also encourage budding software developers to write custom kernels for a variety of operating systems which in turn increases porting of software ROMs from one device to another. This will enable users to install any operating system of their choice on their PCs and phablets.

X. SIMILAR SCHEMES

Google has implemented its Project Treble in its Android Operating System which is designed for the ARM CPU architecture. This facilitates the OEMs to provide faster and uniform software updates across its product portfolio which comply to Project Treble's regulations.

XI. CONCLUSION AND FUTURE SCOPE

A universal interface which enables a software to run on different hardware specifications with the same CPU architecture is discussed. Modular design of drivers is also explained which will be an enhancement to the kernel's memory management. Workflow and operation of the CIB during an I/O operation is elaborated. PPA logic is explained which helps in keeping the CIB chip out of the CPU's control. Minor variations in design and logic which has to be made for different hardware design models is also discussed. Algorithm to apply firmware updates and patches on hardware is explained and comparison between present day's hardware model and the CIB model is also shown. Significance of this technology and the potential changes which might occur after implementation of this technology by hardware vendors are also mentioned.

Future design of the CAUSEWAY INTERCONNECT BRIDGE may include a co-processor which can enable any kernel to run on any CPU architecture. This will further minimize the process of kernel development to a great extent. Further an emulation layer to fix kernel panics [8] occurring in some kernels due to incompatibility of installed hardware may be designed.

REFERENCES

[1] Anon., "*ARM Cortex-A Series Programmers's Guide*", Literature number ARM DEN0013D, pp.10-3, 2014.

[2] J.F. & Ross, K.W., "*Computer Networking: A Top-Down Approach"*. New York: Addison-Wesley. p. 36, 2010.

[3] Nirav Trivedi, Himanshu Patel, Dharmendra Chauhan, "*Fundamental structure of Linux kernel based device driver and implementation on Linux host machine*", International Journal of applied Information Systems (IJAIS), Vol.10, Issue.4, pp.2249-0868, 2016.

[4] Scott Mueller, "*Upgrading and Repairing PCs, Eleventh Edition"*, Que, 2999, ISBN 0-7897-1903-7.

[5] Pete Bennett, EE Times. "*The why, where and what of low-power SoC design*." December 2, 2004. Retrieved July 28, 2015.

[6] Hendric, William (2015). "A Complete overview of Trusted Certificates - CABForum". Retrieved February 26, 2015.

[7] StatCounter, "*Desktop macOS Version Market Share Worldwide Jan 2017 - Jan 2018*" January, 2018.

[8] Steven M. Hancock (November 22, 2002). "*Tru64 UNIX troubleshooting: diagnosing and correcting system problems*", HP Technologies Series, IT Pro collection. Digital Press. pp.119–126. ISBN 978-1-55558-274-6. Retrieved May 3, 2011.

Authors Profile

Mr Pon Rahul M is pursuing his Bachelor of Technology in Information Technology from Loyola-ICAM College of Engineering and Technology and anticipates his degree by 2021. He has hands-on experience on Virtual Desktop Infrastructure. Studying computer technology is both his hobby as well as his passion and he thinks on hilarious ideas during his leisure time, one of which is materialized now. He has performed several experiments which mainly focusses on Kernel Dissection, ROM Porting and software cross-compatibility.

Mr Rishi Shree S is pursuing his Bachelor of Technology in Information Technology from Loyola-ICAM College of Engineering and Technology and anticipates his degree by 2021. He is an inquisitive person with his mind always being curious. He has hands-on experience in Business Process Outsourcing in health-care industry. He is also an avid automobile and tech enthusiast.

