Device driver programming: Implementation of Linux USB Device Driver

Sharanya Mukherjee

Abstract— This paper focuses on the implementation of USB Pen-drive utility program aims at handling the sensible information of operating of modules and kernels where we tend to find out how the module is loaded in the kernel for the execution of the code. The paper also jointly aims at registering the device within the UNIX system. The module where we tend to write the fundamental registration and deregistration program for the USB pen-drive ,has been written in C language . The module file is saved with .o extension within the kernel file.

Index Terms—Device Driver programming, USB device driver, kernel, UNIX, module, Operating system, Linux,

1 INTRODUCTION

The purpose of a device driver is to handle requests created by the kernel with reference to a selected style of device. There's a well-defined and consistent interface for the kernel to form these requests. By noninflected device-specific code in device drivers and by having the same interface to the kernel, adding a brand new device is less complicated. a device driver could be a software package module that resides at intervals the Digital OS kernel and is that the software package interface to a hardware device or devices. A hardware device could be a peripheral, like a controller, tape controller, or network controller device. In general, there's one utility program for every style of hardware device.

2 LITERATURE REVIEW

A device driver, run as a part of the kernel software system, manages every of the device controllers on the system. Often, one utility manages a complete set of identical device controller interfaces. With Digital OS, you'll be able to statically assemble additional device drivers into the kernel than there are physical devices within the hardware system. At boot time, the car configuration software system determines that the physical devices are accessible and useful and might turn out an accurate runtime configuration for that instance of the running kernel. Similarly, once a driver is dynamically designed, the kernel performs the configuration sequence for every instance of the physical device.

As explicit antecedence, the kernel makes requests of a driver by line of work the driver's commonplace entry points (such as the probe, attach, open, read, write, shut entry points). Within the case of I/O requests like read and write, it's typical that the device causes associate degree interrupt upon completion of every I/O operation. Thus, a write supervisor call instruction from a user program might end in many calls on the interrupt entry purpose additionally to the initial invoke the write entry purpose. This can be the case once the write request is divided into many partial transfers at the driving force level. Device drivers, in turn, create calls upon kernel support interfaces to perform the tasks mentioned earlier. The device register offset definitions giving the layout of the management registers for a tool are a part of the supply for a tool driver. Device drivers, not like the remainder of the kernel, will access and modify these registers. Digital OS provides generic CSR I/O access kernel interfaces that enable device drivers to browse from and write to those registers.

Firstly, we should be aware of the fact that whether a driver for a USB device is there or not on a Linux system, a valid USB device will always be detected at the hardware and kernel spaces of a USB-enabled Linux system, since it is designed (and detected) as per the USB protocol specifications. At the

hardware level, the Hardware-space detection is done by the USB host controller which is typically a native bus device The corresponding host controller driver would pick and translate the low-level physical layer information into higher-level USB protocol-specific information. The USB protocol formatted information about the USB device is then populated into the generic USB core layer (the USB core driver) in kernel-space, thus enabling the detection of a USB device in kernel-space, even without having its specific driver available.

3 LINUX COMMANDS USED

A basic listing of all detected USB devices can be obtained using the lsusb command, as root

- lsusb (It is used to view all the usb devices connected)
- lsusb –v (Is is used to give a detailed view of the usb devices
- connected)
- cat /sys/kernel/debug/usb/devices (It also gives detail info of the
- connected devices including the details of interfaces and endpoints)
- sudo –i (in order to log in from root)
- make (to compile a module)
- insmod (inserting a module)
- rmmod (removing a module)
- modprobe (inserting back a module)
- mount (lists all mounted devices)
- fdisk –l (Gives detailed info of the mounted disks)
- mkdir (create new directory)

4 IMPLEMENTATION

The source code is very simple. The header files used are linux/module.h, linux/kernel.h and linux/usb.h. The most important structure is our usb_driver structure. It is the core of our code. It includes 4 main branches. Name, id , prob and disconnect. There are three functions defined inside the structure of usb_driver. Name is simply the name of the device driver.

First of all let's talk about our ID table which is taken in the form of an array. It is because we want to detect as many devices we want. The ID table is fully dependent upon two things. The Vendor ID and the Product ID. Whenever a device is connected this ID_Table function just matches whether the vendor ID and product ID matches to any of the stored ID's. If yes, then it takes control of the device. However , if it doesn't match then it lets go. The vendor ID and Product ID are saved in the USB_DEVICE(vendor,product) function. At the end we have a terminating condition for a situation in which our driver receives null.

Moving on towards the prob function. It is called whenever a device is connected to any of the serial ports in our system. In this function we have two arguments one is the interface pointer while the other is for the USB ID(it is the same as the ID, mentioned above). Other than the arguments I have initialized two different structures. One of type usb_host_interface while other is of type usb_endpoint_descriptor. The host interface has all the information about the interface of the device you have connected. Meanwhile the endpoint descriptor will have the detailed information about the endpoints available in our device. These both are just used in order to print the number of endpoints and to print out the attributes and details of the endpoints. After that we have stored the information of the usb_host into a pointer and then printed the number of endpoints. Then we had a loop from 1 to the number of endpoints available.

Then one by one we have printed the addresses, MXPS and the type of endpoint for all the endpoints inside the device. After that we have a function named disconnect. The name suggests it very well. Yes, the disconnect function is only called when the device is disconnected. We don't have to do anything special in this function. Therefore, I have just printed a statement on the console instead. Now talking about the init function. We all know that init function is the function that is called when you are installing a mod. Therefore, we have registered the module as a usb device driver in the init function. However, if the usb registration is not done properly then the return value will be -1 else than that it will be greater than 0.

To conclude, we can see 5 different lines. The first two lines of module_init(usb_init) is just telling the kernel that the init function for this module is usb_init. Same goes for our favourite exit function. After that we have taken license from GPL. Without this license I cannot call the functions such as usb_register or usb_deregister. It is because the ubuntu has a permission set not letting everyone use those special functions. But when you give a license of GPL then no one can stop you from taking any function from the usb.h library. Next line just shows the author and after that the module_description is showing just a small description of our code.

4.1 MODULE CODE

#INCLUDE<LINUX/INIT.H> #INCLUDE<LINUX/MODULE.H> #INCLUDE<LINUX/KERNEL.H> #INCLUDE<LINUX/USB.H> //PROBE FUNCTION STATIC INT PEN_PROBE (STRUCT USB_INTERFACE *INTERFACE, CONST STRUCT USB_DEVICE_ID *ID) PRINTK(KERN_INFO "[*] OS PEN DRIVE (%04X:%04X) PLUGGED\N", ID->IDVENDOR, ID->IDPRODUCT); RETURN 0 ; } //DISCONNECT STATIC VOID PEN_DISCONNECT (STRUCT USB_INTERFACE *INTERFACE) { PRINTK(KERN_INFO "[*] PEN DRIVE REMOVED\N"); //USB_DEVICE_ID STATIC STRUCT USB_DEVICE_ID PEN_TABLE [] = { //(0781:5567) { USB_DEVICE(0x0781, 0x5567) }, //VENDOR_ID, PRODUCT_ID {} /* TERMINATING ENTRY */ }: MODULE_DEVICE_TABLE (USB, PEN_TABLE); // USB DRIVER STATIC STRUCT USB_DRIVER PEN_DRIVER = .NAME = "MYPENDRIVE", .ID_TABLE = PEN_TABLE, //USB_DEVICE_ID .PROBE = PEN PROBE,.DISCONNECT = PEN_DISCONNECT, **}**: STATIC INT PEN_INIT (VOID) { INT RET = -1; PRINTK(KERN_INFO "[*] CONSTRUCTOR OFDRIVER"); PRINTK(KERN_INFO "\TREGISTERING DRIVER WITH KERNEL"); RET = USB_REGISTER(&PEN_DRIVER); PRINTK(KERN_INFO "\TREGISTRATION IS COMPLETE"); RETURN RET; STATIC VOID PEN_EXIT (VOID) { //DEREGISTER PRINTK(KERN_INFO "[*] DESTRUCTOR OF DRIVER"); USB_DEREGISTER(&PEN_DRIVER); PRINTK(KERN_INFO "\T UNREGISTRATION COMPLETE!"); ł MODULE_INIT(PEN_INIT); MODULE_EXIT(PEN_EXIT); MODULE_LICENSE("GPL"); MODULE_AUTHOR("SHARANYA");

MODULE_DESCRIPTION("USB PEN REGISTRATION DRIVER");

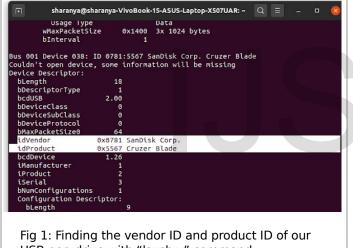
4.2 **KERNEL CODE**

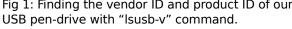
obj-m := usb.o KERNEL_DIR= /lib/modules/ \$(shell uname -r) /build PWD= \$(shell pwd) all: \$(MAKE) -C \$ (KERNEL_DIR) M= \$(shell pwd) modules clean: rm -rf *.o *.ko *.mod.* *.symvers *.order *~

RESULT 5

For this paper, we will be using a SanDisk Pen-drive and write a driver for it. The vendor ID and Product ID can be obtained from the following command-

"Lsusb-v". We can see from the image below that the vendor id for the USB Stick is 0x0781 and Product id is 0x5567.





After successfully building our module we need to insert it using the "insmod" command.

	sharanya@sharanya-VivoBook-15-ASUS-Laptop-X507UAR: ~/Desktop/Modules 🔍 = 💷 🛽
harany	a@sharanya-VivoBook-15-ASUS-Laptop-X507UAR:~/Desktop/Modules\$ sudo make
	password for sharanya:
	/lib/modules/5.4.0-51-generic/build M=/home/sharanya/Desktop/Modules modules
	: Entering directory '/usr/src/linux-headers-5.4.0-51-generic'
] /home/sharanya/Desktop/Modules/usb.o
	ing modules, stage 2.
	ST 1 modules
] /home/sharanya/Desktop/Modules/usb.mod.o
] /home/sharanya/Desktop/Modules/usb.ko
	: Leaving directory '/usr/src/linux-headers-5.4.0-51-generic'
	a@sharanya-VivoBook-15-ASUS-Laptop-X507UAR:-/Desktop/Modules\$ sudo insmod usb.ko
	a@sharanya-VivoBook-15-ASUS-Laptop-X507UAR: <mark>~/Desktop/Modules</mark> \$ dmesg
	000000] microcode: microcode updated early to revision 0xd6, date = 2020-04-27 000000] Linux version 5.4.0-51-generic (buildd@lcy01-amd64-020) (gcc version 9.3.0 (Ub
	3.0-10ubuntu2)) #56-Ubuntu SMP Mon Oct 5 14:28:49 UTC 2020 (Ubuntu 5.4.0-51.56-generic
5.4.65	
	/ 000000] Command line: BOOT IMAGE=/boot/vmlinuz-5.4.0-51-generic root=UUID=d623df23-8bd
	a04c-0f897adfef66 ro quiet splash vt.handoff=7
	000000] KERNEL SUPported CDUS:
	000000] Intel GenuineIntel
	000000] AMD AuthenticAMD
	000000] Hygon HygonGenuine

Fig 3: Inserting the module with "insmod" command.

When the pen-drive is connected it gets registered by the device driver.

FR	sharanya@sharanya-VivoBook-15-ASUS-Laptop-X507UAR: ~/Desktop/Modules 🔍 🗐 – 🛛 🔞
10284.79	5200] usb 1-4: SerialNumber: 20034201301E01F2CC6F
10284.87	8004] usb-storage 1-4:1.0: USB Mass Storage device detected
10284.87	8112] scsi hosti: usb-storage 1-4:1.0
	8206] usbcore: registered new interface driver usb-storage
[10284.88	7732] usbcore: registered new interface driver uas
	7479] scsi 1:0:0:0: Direct-Access SanDisk Cruzer Blade 1.26 PQ: 0 ANSI: 5
	8486] sd 1:0:0:0: Attached scsi generic sg1 type 0
	0075] sd 1:0:0:0: [sdb] 15633408 512-byte logical blocks: (8.00 GB/7.45 GiB)
	0786] sd 1:0:0:0: [sdb] Write Protect is off
	0788] sd 1:0:0:0: [sdb] Mode Sense: 43 00 00 00
	1054] sd 1:0:0:0: [sdb] Write cache: disabled, read cache: enabled, doesn't support
DPO or FU	
	0776] sdb: sdb1
	3157] sd 1:0:0:0: [sdb] Attached SCSI removable disk
	6682] FAT-fs (sdb1): Volume was not properly unmounted. Some data may be corrupt. Pl
ease run	
	8802] usb: loading out-of-tree module taints kernel.
	8892] usb: module verification failed: signature and/or required key missing - taint
ing kerne	
	9475] [*] Constructor ofDriver
	9476] Registering Driver with Kernel
	9528] usbcore: registered new interface driver MYPENDRIVE
sharanya@	sharanya-VivoBook-15-ASUS-Laptop-X507UAR:~/Desktop/Modules\$

Fig 4: Pen-drive gets registered by the device driver

For Deregistering the pendrive and removing the modules we use the "rmmod" command

0 or FUA 2255.930776] sdb: sdb1 2255.930776] sdb: sdb1 2255.930776] sd 1:0:0:0: [sdb] Attached SCSI removable disk 2266.330602] FAT-F5 (sdb1): Volume was not properly unmounted. Some data may be corrupt. P se run fsck. 0502.438082] usb: loading out-of-tree module taints kernel. 0502.43802] usb: module verification failed: signature and/or required key missing - tain 0862.4380475] [*] Constructor ofDriver 0862.439476] [*] Constructor ofDriver 0862.439476] [*] Constructor ofDriver with Kernel 0862.439529] usb:module registered new interface driver MYPENDRIVE 0862.439529] [*] Destructor of driver	sharanya@sharanya-VivoBook-15-ASUS-Laptop-X507UAR: ~/Desktop/Modules 🔍 😑 💷 🏮
225:588460 s674797 scs1:10:00:07 ptrect-Access SanDtsk Cruzer Blade 1.26 PQ: 0 ANSI: 5 225:5886460 scs1:90:00:07 Attached scs1:90:07 sol1:00:00:07 Sol1:00:00:07 225:5900753 sd1:00:00:07 [sdb] 15633408 512-byte logical blocks: (8.00 GB/7.45 CtB) 225:5900763 sd1:00:00:07 [sdb] Mode Sense: 43:00:00 GB/35:5910543 225:5900763 sd1:00:00:07 [sdb] Mode Sense: 43:00:00 GB/35:5910543 225:5900763 sd1:00:00:07 [sdb] Mode Sense: 43:00:00 GB/35:5910543 225:59307763 sd1:00:00:07 [sdb] Write cache: disabled, read cache: enabled, doesn't support 0 or FUA sdb: sdb1 225:39331571 sd1:00:00:07 [sdb] Attached SCSI removable dtsk 225:59307763 rA1:50:00:07 [sdb] Volume was not properly unnounted. Sone data may be corrupt. P 8er run fsck. bslob:00:07 [sdb] Volume was not properly unnounted. Sone data may be corrupt. P 8602.4330802 usb: loading out-of-tree module taints kernel. 8805:4394731 [sdb:stsing - tain 8602.4330802 usb: module vertification failed: signature and/or required key missing - tain 8862:4394731 [sdb:string Driver with Kernel 8862.439473	3206] usbcore: registered new interface driver usb-storage
2255.888486] ad 1:00:010: Attached scst generic sg1 type 0 2255.889676] ad 1:00:00: [sdb] 1563408 512-50tel logical blocks: (8.00 GB/7.45 GLB) 2255.890786] ad 1:00:00: [sdb] Write Protect is off 2255.890786] ad 1:00:00: [sdb] Write Protect is off 2255.930781 ad 1:00:00: [sdb] Write cache: disabled, read cache: enabled, doesn't support D or FUA 2255.930776] sdb: sdb1 2255.933776] sdb: sdb1 2255.933776] sdb: sdb1 2255.933776] sdb: sdb1 2255.933776] sdb: sdb1 2255.933776] sdb: sdb1 2256.933776] sdb: sdb1 2255.933776] sdb2 2255.933776] sdb1 2255.933776]	
2255.0900750 ad 1:0:0:0:0: [adb] 15633408 532-byte ¹ logical blocks: (8.00 GB/7.45 GtB) 2255.0900780 ad 1:0:0:0: [adb] Hode Sense: 43:00 00 00 2255.091054] ad 1:0:0:0: [adb] Wode Sense: 43:00 00 00 2255.091054] ad 1:0:0:0: [adb] Wode Sense: 43:00 00 00 2255.091054] add 1:0:0:0: [adb] Wode Sense: 43:00 00 00 2255.931057] add 1:0:0:0: [adb] Attached SCSI removable dtsk 2265.931057] fd:1:0:1:0: [sdb] Attached SCSI removable dtsk 2265.931057] FAI-156 (adb): Volume was not properly unnounted. Some data may be corrupt. P 8er run fsack sol: loading out-of-tree module taints kernel. 8002.438022] usb: loading out-of-tree module taints kernel. sol: 802.438022] usb: loading out-of-tree module taints digit and/or required key missing - tain 802.438022] usb: loading out-of-tree module taints digit and/or required key missing - tain 802.438022] usb: loading with the mel loading tain loading tain 802.438022] usb: module vertification failed: loading tain loading tain 8	7479] scsi 1:0:0:0: Direct-Access SanDisk Cruzer Blade 1.26 PQ: 0 ANSI: 5
2255.990780] ad 1:0:0:0: [adb] Write Protect is off 2255.990780] ad 1:0:0:0: [adb] Write Cache: disabled, read cache: enabled, doesn't support D or FUA 2255.930776] sdb: sdb1 2255.930776] sdb1 2255.	
2255.090783] sd 1:0:0:0: [sdb] Mode Sense: 43 00 00 00 2255.091063] sd 1:0:0:0: [sdb] Write cache: disabled, read cache: enabled, doesn't support D or FUA 2255.093767] sd 1:0:0:0: [sdb] Attached SCSI removable disk 2256.033062] FAT-f5 (sdb1): Volume was not properly unmounted. Some data may be corrupt. P se run fsck. 2052.438022] usb: loading out-of-tree module taints kernel. 2052.438022] usb: module verification failed: signature and/or required key missing - tain 362.438023] usb: module verification failed: signature and/or required key missing - tain 362.438024] usb: module verification failed: signature and/or required key missing - tain 362.438024] usb: module verification failed: signature and/or required key missing - tain 362.438024] usb:module verification failed: signature and/or required key missing - tain 362.439070] [*] Constructor ofDriver 362.439529] usb:module verification failed: signature and/or required key missing - tain 362.439529] [*] Destructor of driver 362.439529] [*] Destructor of driver	
2025.59:1054] solic:solic: [sdb] Write cache: disabled, read cache: enabled, doesn't support 0 or FUA solic:solic: [sdb] Attached SCSI removable disk 0285.930776] sdb: sdbi void:solic: [sdb] Attached SCSI removable disk 0285.930776] sdb: sdbi void:solic: [sdb] Attached SCSI removable disk 0285.930776] sdb: sdbi void:solic: [sdb] Attached SCSI removable disk 0285.930776] sdb: sdbi: void:solic: solic: solic: 0280.330602] FAT-F5 (sdbi): volume was not properly unnounted. Some data may be corrupt. P 0802.4380802] usb: loading out-of-tree module taints kernel. 0802.438082] usb: module verification failed: signature and/or required key missing - tain 0802.4380470] Registering Driver with Kernel 0802.439520] usbcomi: registered new Interface driver MYPENDRIVE 0802.439520] Registration is conplete 0802.439520] (*) Registration ti conplete	
D or FUA 2255.930770] sdb: sdb1 2255.930770] sdb: sdb1 2255.930770] sdb: sdb1 2256.93062] FAT-FS (sdb1): Volume was not properly unmounted. Some data may be corrupt. P se run fsck. 5062.438082] usb: dodule verification failed: signature and/or required key missing - tain 5062.438082] usb: module verification failed: signature and/or required key missing - tain 5062.438071 usb: module verification failed: signature and/or required key missing - tain 5062.438071 usb: module verification failed: signature and/or required key missing - tain 5062.439470] [*] Constructor ofDriver 5062.439520] usbicmed: registered new interface driver MYPENDRIVE 5062.439520] [*] Destructor of driver	
<pre>255.990776] sdb: sdb1 2255.990776] sdb: sdb1 2255.993157] sd 1:2010:0: [sdb] Attached SCSI removable disk 2265.93062] FAI-f5 (sdb1): Volume was not properly unnounted. Some data may be corrupt. P Sec run fsck. Sec2.438092] usb: loading out-of-tree module taints kernel. Sec2.438092] usb: module verification failed: signature and/or required key missing - tain g kernel Sec2.4380475] [*] Constructor ofDriver Sec2.439476] Registering Driver with Kernel Sec2.439528] usbcom: registered new interface driver MYPENDRIVE Sec2.439529] Registration is conplete [430.69439529] [*] Destructor of driver</pre>	
2255.933157] ad 1:0:0:0: [adb] Attached SCSI removable disk 2265.93062 [FAT-F6 (sdb1): Volume was not properly unmounted. Some data may be corrupt. P 2662.438002] usb: loading out-of-tree module taints kernel. 2862.438022] usb: module verification failed: signature and/or required key missing - tain 2862.439470] [*] Constructor ofDriver 2862.439470] [*] Constructor ofDriver 2862.439470] [*] Constructor ofDriver 2862.439470] [*] Constructor ofDriver 2862.439470] [*] Constructor ofDriver 2862.439520] usbcome: registered new interface driver MYPENDRIVE 2862.439520] [*] Destuctor of driver	
02206.330682] FAT-fS (sdb1): Volume was not properly unmounted. Some data may be corrupt. P se run fsck. 0002.438882] usb: loading out-of-tree module taints kernel. 0862.438882] usb: module verification failed: signature and/or required key missing - tain 0862.439073] [*] Constructor ofDriver 0862.439075] [*] Constructor ofDriver 0862.439072] Registering Driver with Kernel 0862.439072] Registering Driver with Kernel 0862.439072] Registration is complete 0862.439072] [*] Destructor of driver	
se run fsck. 3862.438802] usb: loading out-of-tree module taints kernel. 3862.438802] usb: module verification failed: signature and/or required key missing - tain 3862.438473] [*] Constructor ofDriver 3862.439476] Registering Driver with Kernel 3862.439476] Usbione: registered new interface driver MYPENDRIVE 3862.439529] Registration is conplete 3862.439529] [*] Destructor of driver	
0862.438802] usb: loading out-of-tree module taints kernel. 0862.438892] usb: module verification failed: signature and/or required key missing - tain 0862.439475] [*] Constructor ofDriver 0862.439475] Registering Driver with Kernel 0862.439528] usbcorm: registered new interface driver MYPENDRIVE 0862.439529] Registration is complete 0862.439529] [*] Destructor of driver	
<pre>38624.4380802] USD: module verification failed: signature and/or required key missing - tain 9862.4384753] [*] Constructor ofDriver 9862.439476] Registering Driver with Kernel 9862.439478] USDcom: registered new interface driver MYPENDRIVE 9862.439529] Registration is conplete 9862.439529] [*] Destructor of driver</pre>	
g kernel 0862,439475] [*] Constructor ofDriver 0862,439476] Registering Driver with Kernel 0862,439528] usbcorme: registered new interface driver MYPENDRIVE 0862,439529] Registration is complete 1430.694399] [*] Destructor of driver	
9862.439475] [*] Constructor ofDriver 9862.439476] Registering Driver with Kernel 9862.439528] <mark>Usbcord:</mark> registered new interface driver MYPENDRIVE 9862.499529] Registration is conplete 1930.984399] [*] Destructor of driver	
8862.439476] Registering Driver with Kernel 8862.439528] <mark>usbcore</mark> : registered new interface driver MYPENDRIVE 8862.439529] Registration is complete 1439.894399] [*] Destructor of driver	
8652.439528] <mark>USbCore</mark> : registered new interface driver MYPENDRIVE 8662.439529] Registration is complete 1439.694399] [*] Destructor of driver	
0862.439529] Registration is complete 1439.094399] [*] Destructor of driver	
1439.894399] [*] Destructor of driver	
	1399] [*] Destructor of driver 1484] usbcore: deregistering interface driver MYPENDRIVE

Fig 5: Removing the module with "rmmod" command

We further compile our module using the command "make".



4 CONCLUSION

We have implemented a code for usb device

drivers where we have shown registering of a usb pen drive in linux system. A directory was made which had 2 files that are Kernel and Module file. The module coding has been done in C language and Kernel helps in loading of module for device driver. When the pen drive is connected, it gets registered by the device driver and as soon as we remove it, it gets disconnected and the registration remains incomplete. We conclude that our kernel and module have been properly implemented for the formation of device driver and usb pen drive is getting registered properly.

REFERENCES

- T. Ball and S. K. Rajamani. The SLAM project: Debugging system software via static analysis. In Symposium on Principles of Programming Languages (POPL), pages 1–3, Portland, Oregon, January 2002.
- [2] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. R. Engler. An empirical study of operating system errors. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), volume 35 of Operating System Review, pages 73–88, Banff, Alberta, Canada, October 2001.
- [3] K. Crary and G. Morrisett. Type structure for low-level programming languages. In International Colloquium on Automata, Languages, and Programming (ICALP) 1999, volume 1644 of Lecture Notes in Computer Science, pages 40–54, Prague, Czech Republic, July 1999. Springer Verlag.
- [4] R. DeLine and M. F'ahndrich. Enforcing high-level protocols in low-level software. In Proceedings of the ACM Conference on Programming Laguage Design and Implementation (PLDI), pages 59–69, Snowbird, Utah, June 2001.
- [5] G. J. Holzmann. Software analysis and model checking. In Computer Aided Verification, 14th International Conference (CAV), volume 2404 of Lacture Netra on Computer Science, pages 1, 16, Computer Dependent Luky.
- Lecture Notes on Computer Science, pages 1–16, Copenhagen, Denmark, July 2002. Springer Verlag.

[6] F. M'erillon, L. R'eveill'ere, C. Consel, R. Marlet, and G. Muller. Devil: An IDL for hardware programming. In Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation (OSDI), pages 17–30, San Diego, California, October 2000.

[7] M. O'Nils and A. Jantsch. Operating system sensitive device driver synthesis from implementation independent protocol specification. In Proceedings of Design

Automation and Test in Europe (DATE), pages 562–567, Munich, Germany, March 1999.

[8] S. Thibault, R. Marlet, and C. Consel. Domain-specific languages: from design to implementation—application to video device drivers generation. IEEE Transactions on Software Engineering, 25(3):363–377, May-June 1999.

[9] S. Wang and S. Malik. Synthesizing operating system based device drivers in embedded systems. In Proceedings of the First International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Newport Beach, CA, October 2003.

