11/30/2016

II. LINUX 3.X/4.X DEVICE DRIVER MODULE DESIGN AND IMPLEMENTATION

Programming Project

Karthick Mahalingam (xp9994)
Sachin nagpal (ti6245)

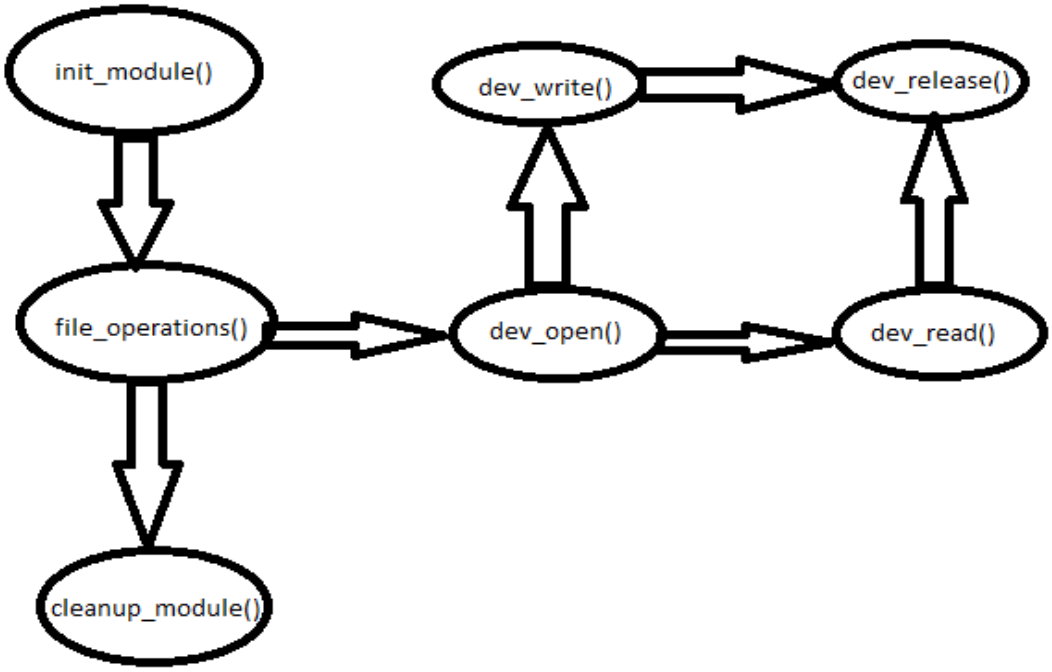
SUMMARY

This project describes a character device driver for Linux OS which performs four basic file operations i.e open , read , write , close and an additional feature of flashing keyboard LEDS. A device driver sits between some hardware and the kernel I/O subsystem. Its purpose is to give the kernel a consistent interface to the type of hardware it "drives". This way the kernel can communicate with all hardware of a given type through the same interface. So basically a device driver is designed and implemented in order to access hardware from a user application so in simple a device driver has basically three sides: one side talks to the user, one talks to the hardware, and one talks to the rest of the kernel.

We also used ioctl operation which is just a minimal kernel module which, when loaded, starts blinking the keyboard LEDs until it is unloaded .In order to run the program, couple of commands used to insert the driver module into kernel ,as soon as module get inserted then init function (which is similar to main) is called for driver initialization , followed by registering a device via function register_chrdev(). After the device gets registered then it will be tied to major number and now user can access the driver module and perform 4 basic functions : open, read, write and close . Once the file operations on device file is performed, then the next task is to unload from kernel space via calling cleanup module to turn off the keyboard LED's.

PROGRAM FLOW:

Below is the pictorial representation of our program flow.



IMPORTANT QUESTIONS:

Linux runs on a variety of hardware platforms. What steps must Linux developers take to ensure that the system is portable to different processors and memory-management architectures and to minimize the amount of architecture-specific kernel code?

Answer: The organization of architecture-dependent and architecture independent code in the Linux kernel is designed to satisfy two design goals: to keep as much code as possible common between architectures and to provide a clean way of defining architecture-specific properties and code. The solution must of course be consistent with the overriding aims of code maintainability and performance. There are different levels of architecture dependence in the kernel, and different techniques are appropriate in each case to comply with the design requirements. These levels include:

CPU word size and endianness These are issues that affect the portability of all software written in C, but especially so for an operating system, where the size and alignment of data must be carefully arranged. CPU process architecture Linux relies on many forms of hardware support for its process and memory management. Different processors have their own mechanisms for changing between protection domains (e.g., entering kernel mode from user mode), rescheduling processes, managing virtual memory, and handling incoming interrupts. The Linux kernel source code is organized so as to allow as much of the kernel as possible to be independent of the details of these architecturespecific features. To this end, the kernel keeps not one but two separate subdirectory hierarchies for each hardware architecture. One contains the code that is appropriate only for that architecture, including such functionality as the system call interface and low-level interrupt management code.

The second architecture-specific directory tree contains C header files that are descriptive of the architecture. These header files contain type definitions and macros designed to hide the differences between architectures.

They provide standard types for obtaining words of a given length, macro constants defining such things as the architecture word size or page size, and function macros to perform common tasks such as converting a word to a given byte-order or doing standard manipulations to a page table entry.

Given these two architecture-specific subdirectory trees, a large portion of the Linux kernel can be made portable between architectures. An attention to detail is required: when a 32 bit integer is required, the programmer must use the explicit int32 type. However, as long as the architecture specific header files are used, then most process and page-table manipulation can be performed using common code between the architectures. Code that definitely cannot be shared is kept safely detached from the main common kernel code.

Describe possible disadvantages of kernel modules. Under what circustances would it be better to keep a kernel split into modules?

There are two principal drawbacks with the use of modules.The first is size: module management consumes un pageable kernel memory, and a basic kernel with a number of modules loaded will consume more memory than an equivalent kernel with the drivers compiled into the kernel image itself. This can be a very signicant issue on machines with limited physical memory.
The second drawback is that modules can increase the complexity of the kernel bootstrap process. It is hard to load up a set of modules from disk if the driver needed to access that disk itself a module that needs to be loaded.
In certain cases it is better to use a modular kernel, and in other cases it is better to use a kernel with its device drivers prelinked. Where minimizing the size of the kernel is important, the choice will depend on how often the various device drivers are used. Where a kernel is to be built that must be usable on a large variety of very different machines, then building it with modules is clearly preferable to using a single kernel with dozens of unnecessary drivers consuming memory. This is particularly the case for commercially distributed kernels, where supporting the widest variety of hardware in the simplestmanner possible is a priority.

What is Device Driver ?

A Driver is just something that provides access to the hardware ,device driver basically allow kernel to access hardware connected to the system

What is kernel module?

Kernel module is a piece of code that can be loaded or unloaded into kernel upon demand.

What is the need and role of device driver?

It is not safe to access hardware linux kernel restrict user space application to use functionality of particular hardware so we need device driver to access hardware with more functionality and flexibility. User activities are performed by set of calls that are independent of driver the way the system call mapping is done to access hardware is role of device driver .

Where is the module code executed ?

Module code is executed in kernel context

How do you locate kernel modules?

Kernel modules can be found in the /lib/modules/kernel-version/directory. Additionally ,you can use the command 'lsmod' to bring up all installed kernal modules.

What are the commands used for compiling and running this project?

insmod ,echo ,cat ,rmmod

How to allocate and freed device numbers?

Register_chrdev_region or alloc_chrdev_region

What are the basic file operations?

There are basically 6 file operations like open , release , seek , read , write , ioctls .

What are the software requirements ?

Linux kernel X.X, GCC, QT.

What are the hardware requirements ?

Intel/amd processor, 128MB RAM

What are the issues faced while running this project?

Pointer initialization and return type errors. Compatibility issues.

Explain the below code block (my_driver->ops->ioctl) (vc_cons[fg_console].d->port.tty, KDSETLED,RESTORE_LEDS);

IOCTL is a function under ops with three acceptable arguments:

File Descriptor, control value and the optional parameter

Name two positive numbers that are used for identification of device files ?

Major number and Minor number

The major device number usually identifies the module that serves the device file or a group of devices served by a module.

The minor device number identifies a definite device in the range of the defined major device number

How can we Free Device Numbers ?

void unregister_chrdev_region(dev_t first, unsigned int count);

What is range of major and minor numbers?

Range is 0-255

What is use of dev_t type ?

This is used to hold device numbers—both the major and minor parts.

What are the primary goals of the conflict resolution mechanism used by the Linux kernel for loading kernel modules?

Conflict resolution prevents different modules from having conflicting access to hardware resources. In particular, when multiple drivers are trying to access the same hardware, it resolves the resulting conflict.

STUDY EXPERIENCES

A lot of the new perspectives and things we have learnt in this Linux device driver programming, we started with our research on deep understanding of procs file system with various reference to grasp the backgroung knowledge of the kernel module like init and release after that allocation of device numbers (major and minor number) then we started the programming part and spent hours on learning code from below references. Finally we made a character device driver with implementation of four basic file operation i,e open, read ,write and close as per the project requirement. We initially faced program compatibility issues like pointer initialization, header file etc but managed to fix with the help linux metalink and integrated LED blink feature basicall using ioctl operation on a keyboard driver. We researched further why the LED light are used for CAPS Lock notifications and not for any other purposes and studied the below pros and cons:

1) Separate scheduling is needed for the usage of LED on a muli-processor system as multiple process runs on a system.
2) Ioctl operations can be used to access other driver and its hardware irrespective of the vendor

References :

[1] http://kevinboone.net/linux kernel file 0.html

[2] https://static.lwn.net/images/pdf/LDD3/ch16.pdf

[3] http://ytliu.info/notes/linux/file ops in kernel.html

[4] http://iacoma.cs.uiuc.edu/ nakano/dd/drivertut3.html

ANALYSIS

This project can be analysed into 2 parts : one is a character device driver-module and other part is flashing keyboard LEDs. The overall project performs five operations i.e open , read , write, close and ioctl. ioctl is a minimal kernel module which, when loaded, starts blinking the keyboard LEDs until it is unloaded and charcter device driver performs all file function when module is loaded so basically we can say a device driver has three sides: one side talks to the user, one talks to the hardware, and one talks to the rest of the kernel .

EVALUATION

This project provides the functionality of both character device driver and Flashing keyboard LED. Character devices, the simplest of Linux's devices, are accessed as files, applications use standard system calls to open them, read them, write to them and close them exactly as if the device were a file. As a character device is initialized its device driver registers itself with the Linux kernel by adding an entry into the chrdevs vector of device_struct data structures.

There were some functional dependencies that are based on kernel version and Linux header files .It means kernel codes are neither upward compatible nor downward compatible .We tried to run project on 2 different laptops , one of dell with kernel version 4.3 and another laptop of Lenovo with different kernel version . It failed and gave lot of errors like pointer initialization , then it was fixed by adding header files. We also found that the program lacks multiple device access.There is a chance of device file access conflicts when it is accessed by two kernel program. The kernel program should be mutually exclusive they both access the same device file

FUTURE DEVELOPMENT

Our future Development of the project is to achieve the below task:

1) Integrate sound module with this flashing keyboard LED and make it available to the multiple resources and to use it as a notification or an alarm until intended operation is completed. Make this LED module accessible to the user space program .
2) Planning to design a mechanism on how to provide access of LED to multiple process effectively
3) Make the character device driver kernel program mutally exclusive when multiple module access the same device file using semaphore.
4) Make use of fuse filesystem for the device file-(The **Filesystem** in Userspace (**FUSE**) is a special part of the Linux kernel that allows regular users to make and use their own **filesystems** without needing to change the kernel or have Root privileges)