Introduction to Kernel & Device Drivers

Raj Kumar Rampelli

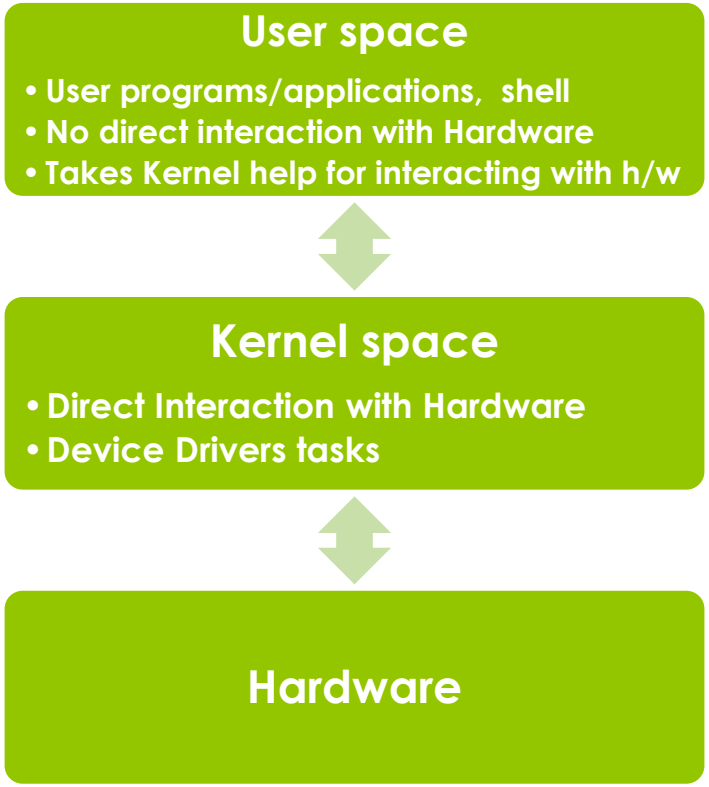
Outline

- What is kernel ?
- What is a Device Driver ?
  - Types of Device Drivers
- What is a Module ?
  - How to work with Module
- Platform devices and its registration
- How does probe() gets called ?
- Role of Vold in volume mount
- Allocation of memory in Kernel
- References

What is Kernel ?

- Kernel is a piece of code & core part of operating system and whose responsibilities are
  - Process management
  - Memory management
  - Device Input/output control: Device Drivers task
  - File system
    - Everything is a file in the linux including Device connected to system
- Kernel code is written in C and assembly language
- Kernel code always runs in Kernel Space

Kernel Space & User Space



RajKumar Rampelli

What is Device Driver ?

- Piece of code that will be executed when suitable device is connected to the system
  - Each device has its own code in the operating system
  - This code is called "Device Driver"
  - Works for a specific h/w device
- Device Drivers
  - Hides the details of how a device works
  - Bridge between user applications and Hardware
    - Mapping user application calls to specific h/w
  - Support Hot-plug feature
    - Drivers will be loaded when needed & will be unloaded when not needed
  - Also called as "Modules"

Types of Device Driver

- Character device driver
  - Byte oriented data transfer
- Block device driver
  - Block(512Byte) oriented data transfer
- Network device driver
  - Packet data transfer

What is Module ?

- Loadable Kernel object – contains related subroutines, data grouped together
- Module executables are represented in kernel object format (.ko files)
- It has only one entry function & one exit function
  - Entry function: called when loaded into kernel
    - Initialization function (like main() in C)
    - init_module(void)
  - Exit function: called when removed from kernel
    - cleanup_module(void)
- Enables hot-plug feature
  - i.e. dynamic (runtime) loading/removal of drivers into/from kernel in response to device state

Working with Module

- Insert/load a module into system kernel
  - *insmod module_name.ko*
- Remove/unload a module from system kernel
  - *rmmod module_name.ko*
- Insert a module along with dependent/related modules in one go.
  - *modprobe module_name.ko*
- Detailed explanation on Module is available at http://practicepeople.blogspot.in/2013/06/kernel-programming-1-introduction-to.html

__init* macro() usage

- Syntax:
  - *__init* function_name();
  - *__initdata* variable_name;
- Compiler stores the above function/variable in "init.text"/"init.data" section in vmlinux.ids file
- Tells the compiler to free the memory (*free_initmem()*) once the task of function/variable associated with init macro is completed.
  - free_initmem(): memory clean up task
  - Memory optimization
  - Check Kernel log for "*Freeing unused kernel memory: xxxk freed*"
- Where to use this macro:
  - Module 's Initialization function, since it is called/used only once when loading the module (or during kernel boot up)
- More info can be found at http://practicepeople.blogspot.in/2013/07/kernel-programming-3-init-macros-usage.html

Device registration and initialization

- Platform devices
  - Devices that are integrated into a given chip and therefore are always there
  - The platform-specific initialization code statically initializes such arrays of platform_devices and then registers them using platform_register()
  - Therefore there is no need for sophisticated probing.
    What is probing()[Discussed in the next slide]
  - Instead, the string contained in platform_device.name is compared platform_driver.driver.name and a match is assumed if they are equal.

How probe() gets called ?

Probe() is called when device is recognized by the platform

Driver's init function gives kernel a list of devices it is able to service, along with a pointer to a probe function. Kernel then calls the driver's probe function one for each device.

- probe function starts the per-device initialization:
  - initializing hardware,
  - allocating resources, and
  - registering the device with the kernel as a block device.
- ***Example:*** *Kernel/drivers/mmc/host/sdhic-tegra.c → sdhci_tegra_probe()*
- Host controller initialization at *kernel/drivers/mmc/host/sdhci.c → sdhci_add_host()*
- Device initialization starts in *kernel/drivers/mmc/core/core.c → mmc_rescan()*
  - Starts execution when Host detects the device.
  - mmc_attach_sdio() → sdio.c [core driver]
  - mmc_attach_sd() → sd.c [core driver]
  - mmc_attach_mmc() → mmc.c [core driver]

Vold (Volume Manager Daemon)

- Listens to kernel Netlink socket events for volume status change
- Interacts with MountService (Java layer)
  - Decision maker to mount any media/device
  - Receives volute state change notifications from Vold
  - Uses Unix domain (POSIX Local IPC) socket to communicate with Vold
- Ex: Insert SD card on Android device
- It stores volume information retrieved from vold.fstab file
  - This file describes what storage devices will be added into the system. Its format:
    <mount> <volume label> <mount point> <partition#> <sys fs path to the device>
- Netlink Socket: Pass the information between Kernel and User space
- Unix domain socket declare: $TOP/system/core/rootdir/init.rc
  socket vold stream 0660 root mount

Allocation of memory in Kernel

Kmalloc()	Kzalloc()	Kcalloc()	Vmalloc()
Allocates contiguous physical blocks of memory in byte-sized chunks	Allocates contiguous physical blocks of memory	Allocates memory for an array	same as kmalloc, except it allocates memory that is only virtually contiguous
Memory is not set to zero	Memory is set to zero	Memory is set to zero	underling physical memory can be discontiguous
void * kmalloc (size_t size, int flags)	void * kzalloc (size_t size, int flags)	void * **kcalloc** (size_t *n*, size_t *size*, unsigned int __nocast *flags*);	void * vmalloc (unsigned long size)
Depends on the flag used, it may sleep	Depends on the flag used, it may sleep	Depends on the flag used, it may sleep	Can sleep, so don't use it in interrupt context

Allocation of memory in Kernel

Kmalloc()	Kzalloc()	Kcalloc()	Vmalloc()
Can allocate upto 4MBytes	Same as kmalloc	NA	to obtain very large regions of memory
Simple and fast	Simple, preferable and fast	Simple and fast	Slow compare to kmalloc

Vmalloc(): Slower and not advisable if required memory is less. Because,

1. Makes non-contiguous physical memory to continuous virtual memory
2. Setting up the page table entries
3. Pages obtained from vmalloc() must be mapped to their individual pages since physical memory is not contiguous
4. Results in much greater TLB, performance is affected.

Note: **malloc()** function also works in same manner. i.e. allocated memory in continuous in virtual address space of the processor, but there is no guarantee that they are contiguous in the physical address space (physical RAM)

Allocation of memory in Kernel

- Flag: controls the behavior of memory allocation and divided into 3 groups
  - Action modifiers: How to allocate memory. Ex: can sleep or not
  - Zone modifiers: Where the request should be satisfied. (DMA buffers)
  - Types: types of allocation

FLAG Type	Action Modifier	Zone Modifier
GFP_KERNEL: use in process context, safe to sleep	(__GFP_WAIT | __GFP_IO | __GFP_FS)	
GFP_ATOMIC (High priority & Does not sleep). Use in Interrupt handlers, bottom halves where kernel should not sleep	__GFP_HIGH	
GFP_DMA		__GFP_DMA

__GFP_HIGH: The kernel can access emergency pools.
__GFP_WAIT: The kernel can sleep
__GFP_IO: The kernel can start disk I/O
__GFP_FS: The kernel can start filesystem I/O
__GFP_DMA: Allocate only DMA-capable memory

References

- Linux Journal Article: http://www.linuxjournal.com/article/6930
- Third Edition of **Linux Device Drivers**, by Jonathan Corbet
- Vold topics in slideshare.net

THANK YOU ☺

Have a look at

My PPTs: **http://www.slideshare.net/rampalliraj/**

My Blog: **http://practicepeople.blogspot.in/**