1044414: Advanced Operating Systems and Virtualization **Spring 2018**

Lecture 11: April 13

Lecturer: Alessandro Pellegrini *Scribe: Anxhelo Xhebraj*

To register char devices there are some functions. You have to give a name to the device and specify the types of operations. If the major number is put to 0 then you ask the kernel to give some available major number. To unregister you must specify the major number and the name. Unregistering doesn't mean to remove the device but means to unmount the modules for that device.

Also for char devices we need file operations. The owner is the module that drives the devices.

register chardev and alloc chrdev allow to identify the major number and minor number from which to start. In the first case you want a specific range of minor numbers therefore if one module is using one of them the call is going to fail. In the second case the kernel is going to scan for the availability of minor numbers and return the starting point.

Block devices behave differently. In the last release block devices apis are exposed in genhd. The struct is gendisk and before it was called genhd. The struct describes a major, minor and minors (you can specify a range). Given a major you specify the maximum number of minors it can support i.e. the maximum number of partitions. request queue: the block device returns a block of data while the char device a stream. Since the former might take time the fops and queue are managed such that the scenario is controlled.

Block device ops are slightly different from file operations: we don't have anything for reading or writing a block device. Since it may take time to do those ops another strategy is used instead of a specific function pointer.

There might be a plethora of requests waiting for a device and the developer of the module doesn't have an overview of it therefore another strategy is used.

Request queues are the way to operate on block devices. Any time a process is put to sleep for waiting to some device the request queue is used.

The struct inode there is a field called imode that tells the type of the inode. mknode creates a generic i-node. If the inode represents a device the kernel is going to look in the device database to findout the actual device that implements those operations. irdev is used in case imode tells that represents a char or block device to know the major and minor number.

mknod syscall, given a pathname, mode and dev can be used to create some kind of node. Regular, char, block, pipe. dev is used only if block or char is the node type.

umask enables to specify some permissions of the process/users

for interacting with char devices the linux kernel uses a generic function called chrdev open. Open is mapped to vfs open that which will perform some pathname lookup to find the inode, will find out that it is a char device and then handouts the opening to the char devices subsystem through chrdev open. the function will issue a call to a function called kobject lookup, allows to find some specific object in the kernobj subsystem. From the kobj we can navigate to the cdev and finally to ops. Since this full walk is costly the pointer to ops are cached in the inode. Therefore if we would like to dynamically change the pointers of our fops we cannot do that because of caching. Therefore we do a switch case in a generic function pointer that will multiplex the ops depending on something.

source is kind of the partition `/dev/sda1`. As soon as we connect a device that device will appear in `/dev` mount target is the target vfs where we want to mount this fs. the type is the one of the source, mountflags specifies how we want to mount the filesystem for example saying if we want it to not be executable. We can remount to for example change from rdonly to write.

A mount point is identified through the member dflags in dentry. DCACHE MOUNTED specifies that dentry is associated to a mount point and not to any file. The lookup function skips dentries of this kind.

RCU inside the PCB. RCU (Read Copy Update). The counter in the file descriptor table tells how many entities are using the filesstruct. struct fdtable is the actual file descriptor table and a pointer to the same table. Therefore an actual field and the pointer to the field. You take the pointer which is the consistent state (maybe older) of the filesystem. The directive rcu tells the compiler how to use the pointer. There is a spinlock for doing stuff and it is cached aligned. next fd tells the kernel where to start to look for an available file descriptor.

fdtable contains two bitmaps. Once you fork and exec you inherit descriptors by default. To disallow some inheritance of the file descriptors the close on exec is used. The process must specify the flag when opening a file. Open fds speeds up the lookup for available file descriptors. Since there might be portions of the openfds bitmap that are full of 1s then the fullfdsbits contains 1 and 0s for each group of bits in openfds.

```
openfds 10010010 11111111 00110011
fullfds        0        1        0
```

file struct. fpos keeps track the position of the cursor telling where the last read has arrived and the lseek. fcred keeps credentials to specify different capabilities of the users to manage the file.

How is a file opened? open eventually calls dosysopen and then a set of additional calls. You must have an available file descriptor and then the struct file is allocated. In the second part you ask the vfs to do the actual opening of the file: do filp open. This call will return the struct file associated to the file.

In the pcb there is a namei ds (pointer) associated to the current path in which the process is living (different from working directory).

The pathopenat gets a file descriptor fileld of zeros through get empty filp (taken from slab).

IS ERR and PTR ERR allows to use a pointer for returning an error code.

get unusued fd flags. Through files fdtable you get a snapshot of the file descriptor table.

must check tells the compiler to generates warnings if the call to the macro doesn't check the return value.

The definition o a syscall if wrapped in syscall define macros. Similar to the generation of stubs in userspace. This boils down to an asmlinkage defined function. closefd releases the file descriptor and invokes the closing function and discards the ds not needed anymore. eventually it calls put unusued fd, the data structure is not used anymore. filpclose findsout the file operation to be executed to close the file and decrement the reference counters into dentries and inodes evenutally putting them.

donitify flush disposes the dentry. Remove the locks in the struct file.

close fd

put unusued. clear sets bit to 0 in the bitmap, next fd is put to the one one now empty.

write

vfs write is a wrapper to the actual write. Security checks on the pointer and then call the underscore version.

read syscall, get fd, check if file associated, check the position etc.

11.1 Proc file system

In memory filesystem that exposes informations to the user about processes running in the machine.

proc dir entry used to simplify the management of proc opearations which are very specific.

11.2 Sysfs File System

Creates a mapping btw kernel objects and file system. Used also to pass parameters to kernel modules.

ksests is a data structure embedding kernel objects.

In the early days of linux when inserting a device the entry in `/dev` would appear. At some point they gave the control to create the entries to the user space programs. The kernel module only setups some kernel objects.

udev listens for new files in `/sys`. dbus for notification.

The driver must create the kobj for sys otherwise it won't be mounted.

References

[a20()] A20 line. URL https://wiki.osdev.org/A20_Line.

[car()] Writing a bootloader from scratch. URL https://www.cs.cmu.edu/~410-s07/p4/p4-boot.pdf.

[con()] Context switching. URL https://wiki.osdev.org/Context_Switching.

[des(a)] Descriptor cache, a. URL https://wiki.osdev.org/Descriptor_Cache.

[des(b)] Interrupt descriptor table, b. URL https://wiki.osdev.org/Interrupt_Descriptor_Table.

[osd()] "8042" ps/2 controller. URL https://wiki.osdev.org/"8042"_PS/2_Controller.

[rma()] The workings of: x86-16/32 realmode addressing. URL https://web.archive.org/web/20130609073242/http://www.osdever.net/tutorials/rm_addressing.php?the_id=50.

[ser()] Interrupt service routines. URL https://wiki.osdev.org/Interrupt_Service_Routines.

[tas()] Task state segment. URL https://wiki.osdev.org/Task_State_Segment.

[vec()] Interrupt vector table. URL https://wiki.osdev.org/Interrupt_Vector_Table.

[wra()] Who needs the address wraparound, anyway? URL http://www.os2museum.com/wp/who-needs-the-address-wraparound-anyway/.

[x86()] Why doesn't linux use the hardware context switch via the tss? URL https://stackoverflow.com/questions/2711044/why-doesnt-linux-use-the-hardware-context-switch-via-the-tss.

[wik(2017)] Task state segment, Jun 2017. URL https://en.wikipedia.org/wiki/Task_state_segment.

[wik(2018a)] x86 memory segmentation, Mar 2018a. URL https://en.wikipedia.org/wiki/X86_memory_segmentation#cite_note-Arch-1.

[wik(2018b)] A20 line, Feb 2018b. URL https://en.wikipedia.org/wiki/A20_line.

[Bovet and Cesati(2006)] Daniel P. Bovet and Marco Cesati. *Understanding the Linux kernel*. OReilly, 2006.

[Brouwer()] Andries E. Brouwer. A20 - a pain from the past. URL https://www.win.tue.nl/~aeb/linux/kbd/A20.html.

[Collins(a)] Robert Collins. A20 - reset anomalies, a. URL http://www.rcollins.org/Productivity/A20Reset.html.

[Collins(b)] Robert Collins. The segment descriptor cache, b. URL http://www.rcollins.org/ddj/Aug98/Aug98.html.

[Duarte(2008a)] Gustavo Duarte. Memory translation and segmentation, Aug 2008a. URL https://manybutfinite.com/post/memory-translation-and-segmentation/.

[Duarte(2008b)] Gustavo Duarte. Cpu rings, privilege, and protection, Nov 2008b. URL https://manybutfinite.com/post/cpu-rings-privilege-and-protection/.

[Intel()] Intel. *Intel 64 and IA-32 Architecture Software Developer's Manual*, volume 3A.

[Oostenrijk(2016)] Alexander van Oostenrijk. Writing your own toy operating system: Jumping to protected mode, Sep 2016. URL http://www.independent-software.com/writing-your-own-toy-operating-system-jumping-to-protected-mode/.