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

Lecture 10: April 10

Lecturer: Alessandro Pellegrini *Scribe: Anxhelo Xhebraj*

Init has to read from disk the configuration files and determine the service related to this target. In unix the virtual file system (VFS) allows to read such file. In unix everything is a file: pipes, devices, etc. The VFS abstracts everything related to driving the devices. There is one representation in RAM of the filesystem either partial or full structure of the file system. The representation in RAM allows to navigate in a fast manner the filesystem. Acts as cache for data or metadata about what is saved in secondary storage. The filesystem independent part is the interface for working with files while the dependent part is about using drivers etc.

Every FS object is represented via specific data structures in RAM. Usage of function pointers allow to access different devices through the same interface.

File system is complicated in terms of data structures. In start kernel it is initialized. init rootfs initializes the ram filesystem. The ram fs will mount the root fs and unload the ramfs. The virtual fs is so modular that you can run linux without fs except the parts of the vfs for accessing some devices.

Since we're dealing with multiple kinds of file systems we must know how to identify a specific file system type and deal with the data associated to it. A structure called file system type that gives info about the name of the filesystem. The kernel will resolve then the mounting through the name. read super is a function pointer associated with what the kernel should execute when mounting that device. The kernel will look into the partition table to know the structure and through the type name will use. One can register a filesystem. In the usb or device is written the type of filesystem stored.

rootfs type (global variable) is registered by init rootfs. rootfs mount is specific for the rootfs and is the function executed for mounting the filesystem while kill sb is the function for unmounting and this one is generic.

The monting of the rootfs is performed in init mount tree. Four ds are involved: inode vfsmount etc. This datastructure keep information of various portions of the fs. They have pointers and they use pointer to each other as a graph. vfsmount and superblock describe the attributes of a specific instance of the filesystem.vfsmount describes relations across different filesystems for example when mounting some filesystem into another filesystem. It also knows that when unmounting some filesystem you cannot since another fs is linked in a lower level of the hierarchy.

vfsmount is the organization of the 3.0 kernel. mnt parent is the pointer to the parent of the current vfsmount.

dentry stands for directory entry or dirent. This is a cache, information about some elements of a path independently of the physical representation of the system. Represents the root of the mounted tree. superblock keeps information about how to manage the fs as a whole. A couple of lists used to enter childrens. A counter tells how many times the fs has been mounted. How many instances are in the system: one can mount the same fs on multiple points. Flags. Strings are used for the name of the device.

superblock keeps a pointer to file systme type. A pointer to dentry which keeps the root of

the filesystem. A list of dirty inodes to know what data should be written back to memory.

The super operation pointers is a struct of pointers telling how to manage this block.

dentry is the ds used to navigate the filesystem. From it we can go to the inode to access the data on disk. inode keeps the pointer. dparent is the only parent of the dentry. a set of different childrens are kept. dname keeps the full name while diname keeps the short name. lockref is a structure that keeps both a lock and a reference counter (atomically increased and checked whenever tried to be modified). A pointer to the superblock it belongs. Multiple lists are used for dealing with concurrent access.

qstr used for long names is a quickstring. A pointer to some buffer where the string is located. What happens when we need to do a strcmp btw qstrings? too expensive, hash used instead to speedup the comparison.

inode organization: is the companion of the dentry. Here we need to differentiate the code used to manage the inode. Inodeops keeps a set of function pointers which specify what to do for spacific operations (ram representation of the node). fileops used for managing the data in disk through drivers. waitqueue used to put to sleep processes when it needs to wait for some device for performing some tasks.

Logically every filesystem can be mounted on any dentry.

Back to the initialization of the root filesys. It allocates the four ds needed and link them. Sets the name / to the rootfs and links the idle process to rootfs. Every process lives in a specific directory of the fs callent current process directory. The idle process is linked to rootfs which is the only fs living on system startup.

`init_mount_tree` at a certain point calls vfs kern mount that mounts the rootfs. current is a reference to the process/kernel thread that is currently running in the current processor.

Let's look to fs in the Process Controle block. At any time one must know the actual process working directory and the root folder. fs struct has two pointers pointing to the root node fs and the working directory. In kernel 3.0 fs struct is modified: count and lock are changed to a spinlock and a sequence counter. Different way to deal with concurrency in the kernel. Rewrite of the concurrency from coarse grained to finer grained. the other entries are moved to root and pwd that speedup some stuff and solves issues not specified in posix. Problems with symlinks which are elements in the fs that point to different elems in the fs. the two entries might belong to different vfsmounts and since the latter is not linked to the dentries we must put the pointers.

superblock ops allow to manage statistics, create and manage inodes. some fsops might just not do nothing for example ramfs. The organization is in struct super operations. One function pointer for each op that must be supported in the fs. put inode means that you don't need anymore therefore it can be disposed. put super, dispose the superblock (usually done for unmounting). These operations are for ram operations. destroy forces the other threads to stop using that inode.

Ramfs is easy and doesn't need to actually drive any device.

dentry ops ddelete decrements a counter and if it reaches 0 it also puts it.

inode ops allows to creat, link, unlink etc. Lookup op is important since it tells that i'm looking the fs to find something. you can have inodes for different things for example for a file or device. mknod creates a generic inode.

inodesops in the ramfs: lookup has nothing strange. Some functions are generic others are specific for the ramfs.

nameidata.c is a subsytem into the VFS subsystem that takes care of handling names meaning some path in the vfs. Taking care of the different natures. This allows to lookup some elements in the fs abstracting and taking care of the different elements in the VFS. It is used to manipulate strings for paths and perform lookups.

10.1 VFS Intermediate Functions

`path_lookupat` lookups some elements of the filesystem. nameidata is used to return information. It does a tree walk through `walk_component` used to reach the latest component. Flags tells whether to follow symlinks and other stuff. struct path allows also for relative paths.

vfs mkdir creates a new inode. dir points to the parent inode and modes specifies the access rights. dget dentry acquires a dentry and increments the counter. dput tells that the dentry is not needed decrementing the reference count. If it goes to 0 it is removed from memory. Recursively is checked if also the parents are needed anymore. domount mounts a device into a target directory.

dhash and lookup is introduced in newer versions of the kernel. Tells the qstr is a new structure so the hash function is not computed yet. this function will hash it and will rely on dlookup. dlookup expects a hash.

vfscreate creates an inode related to some dentry.

VFS functions are used by kernel modules if you're not reimplementing the fs. With the other functions you might mess the ram representation of the filesystem and maybe also disk.

kernel init will initialize the vfs and the execution of init. do basic setup initializes drivers etc. prepare namespace waits for all devices to complete the startup. Mounts the dev pseudofolder and will load also the initramfs into kernel memory from ram and finally mounts it in /.

run init process will perform many try to run init process through the execve.

How does linux manage devices? a device has a number associated to it. In old versions from 0 to 255. These were actually associated to the family of devices. Indeed linux uses MAJOR and MINOR number. the former is used to access a driver database to know how to interact with that device.

Linux sees each partition as a different devices. But they belong to the same family and therefore have the same MAJOR number because it is accessed in the same manner but the different instances are differentiated through the MINOR number. The minor number is assigned by either the kernel or driver.

There are two kinds of devices mainly: char or block. The former can return any number of bytes while the latter return a fixed number of bytes.

The kernel keeps the info to know the type of device.

family: char or block then within the family the major number.

10.2 Device Database

cdev and bdev maps are the databases keeping the information about the devices. In kernel 2.6 kernel objects were defined. There can be 255 elements for each database. Each database is protected by a block. This is implemented as an hashmap and accessed through

the major number. module points to the driver to manipulate that device. data points to a datastructure.

cdev struct describes character devices. This is the one pointed by data. The first element is kobject. A pointer to file operations that tell how they are implemented for the specific device and a list head since it can be chained somewhere. randomize layout enables to take the members of the struct and shuffle them.

kobject is a generic object that has a name, can belong to a list or a set. Has a specific type and can be related to the sysfs.

list head in kobject points to the cdev.

with char devices you might have some driver wanting to reserve some minor numbers. For each major you can define a set of minors from baseminor to baseminor + minorct.

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/.