Banat's University of Agricultural Sciences and Veterinary Medicine from Timisoara

Contact: web: *http://dse.usab-tm.ro*, e-mail: *bjb@usab-tm.ro*

ANALYSIS OF VIRTUAL FILE TYPES USED BY THE LINUX OPERATING SYSTEM

Cristian Vasile

University of Craiova, Faculty of Agriculture
e-mail: cristi_vasile_4you@yahoo.com

Abstract. The rapid evolution in recent years in all areas of activity has led to the widespread use of computers and automatic processing technologies of data and information. By connecting to the Internet there is also the possibility of an exchange of information between users of the network. For these reason a very important problem for any user of the computer is the security of the dates and the LINUX operating system can to offer this pressing requirement.

This article proposes a study on virtual file types that can be used by the LINUX operating system. Through this manner of implementation the LINUX operating system can to use some different types oh file systems: XFS, EXT2, EXT3, JFS and Reiser. For that reason, in the present paper the capacity of LINUX operating system through the mediation of the virtual file system to offer the support for this types of file systems is analyzed.

Key words: file system, structure, virtual, operating system, process, inode.

Introduction

The operating system means a software package that allows a human operator to enter into a "dialog" with the computer, to perform the desired operations.

Throughout evolution of computing systems have been implemented and has been used several operating systems, such as: MS-DOS, Windows, Linux, Unix, etc.

In recent years we can observe an increasing trend of computer users to use the Linux operating system to the detriment of the known Windows and it's very primarily due to increased safety in the use of computer and information technology applications.

At first, the file system used by Linux operating system was the appropriate to MINIX, which was characterized by some of the limitations with regard to the length of the file name with a maximum of 14 characters and also any limitations on the size of the files to a maximum of 64 MB.

Given these characteristics of the initial implementation, there have been some researches to design more efficient file systems [MUŞATESCU, 1999, NICOLAE, 2004].

Thus, the first file systems with improved performance was Ext, which allowed saving files whose names can be written up to 255 characters and the size of the files as they work to be up to 2 GB of memory [MUŞATESCU, 1999, NICOLAE, 2004].

As a result of research conducted has been tried and has failed to eliminate inconveniences caused by the more work time and has been implemented Ext2 file system, this being the most important file system used by Linux.

Also, in the desire of specialists to offer new features for working with Linux operating system have been implemented and other file systems, of which we can mention: Ext3, JFS, XFS, Reiser and virtual file system [NICOLAE, 2004].

Theory

Virtual file system used by Linux

As already explained, the Linux operating system can work with several types of file systems. Thus, to provide support for working with different file systems, Linux operating system contains a special interface to the kernel, called Virtual Filesystem Switch (VFS interface). Figure 1 shows how works this interface VFS [NICOLAE, 2004, TANENBAUM, 1999].

Thus, when the link operation is performed in Linux, may be able to choose the file system to be built into the kernel.

Other types of file systems can be loaded dynamically, at the request of the user, as distinct modules during the execution of commands you want.

Available on-line at http://dse.usab-tm.ro
Banat's Journal of Biotechnology
2011, II(4),

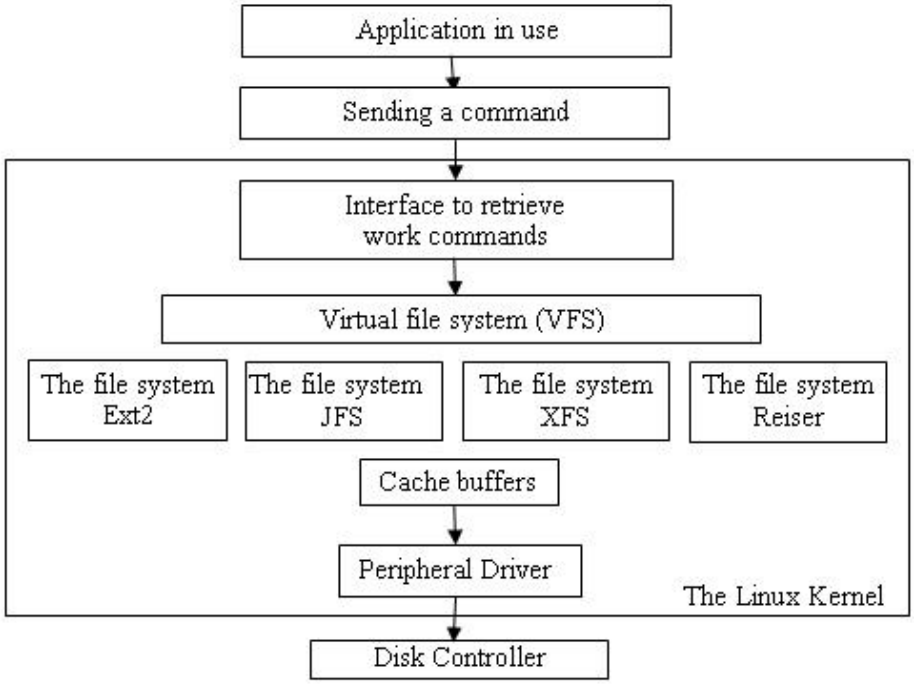

Figure 1. The workings of the VFS interface

Cache of inodes

To work with the virtual file system in Linux, by file fs/inode.c was implemented properly the cache of inodes.

As a general structure, the cache consists of a:

a) global hash table, called inode_hashtable, where each node can be found in search pointer value to his block and an inode number stored in 32 bits. Inodes without a corresponding block will be linked in a double-linked list and the top of this list will be stored in anon_hash_chain double.
b) global list named inode_in_use, which will contain all the valid inodes (usable) is characterized by positive values of the corresponding parameters (i_count > 0 and i_link > 0).
c) global list named inode_unused, which will contain all the inodes with the parameter i_count = 0.
d) list maintained for each block, indicated by the sb→s_dirty, which contains the valid inodes with values parameters i_count>0, i_nlink>0 and i_state&i_dirty=TRUE. Keeping the memory of such lists allow rapid synchronization of inodes.
e) massive cache of corresponding inodes called inode_cachep. When inode type objects are allocated or activities are issued, they will be taken from the massive that returned to this massive of inodes.

It should be noted that all these lists that are linked through field inode→i_list, are protected by inode_lock field [NICOLAE, 2004].

File management descriptors

The descriptor of a file means a distinct parameter who is assigned at this file, which distinguishes it from all other files that the Linux operating system working at a time.

Between the corresponding file descriptors of applications in work and structures of inodes maintained by the kernel of Linux operating system, there are several levels of indirection [NICOLAE, 2004].

Thus, when a process starts a command open(), the kernel will immediately return a positive integer, which will attach the file that is currently in working and will be used in operations of input/output (I/O) that will perform on that file.

The positive integer generated by the kernel will behave as an index into an array of pointers to structures of type file. Such a structure contains a field f_dentry that points

to a field dentry and each dentry points to an inode through the field named d_inode.

The task_struct structure corresponding to each application that runs under the Linux operating system contains a field named files, that point to the files_struct structure who is defined in the file/include/linux/.sched.h.

The pointer task_struct→files corresponding to each application in working points to the task_struct structure, which in turn implements the database table of open processes (called TDFU), whose syntax is as follows:

```
struct files_struct{
        atomic_t count;
        rwlock_t file_lock;
        int max_fds;
        int max_fdset;
        int next_fd;

struct file **fd;
/* current vector of file descriptors fd */
        fd_set *close_on_exec;
        fd_set *open_fds;
        fd_set close_on_exec_init;
        fd_set open_fds_init;
struct file *fd_array[NR_OPEN_DEFAULT]}
```

The field named count of each structure of type file implements the reference counter of that file.

File structure management

For the Linux operating system, the structure of the file type is declared in the file /include/linux.fs.h as:

```
        struct file {
                struct list_head f_list;
                struct dentry *f_dentry;
                struct vfsmount *f_vfsmnt;
                struct file_operations *f_op;
                atomic_t f_count;
                unsigned int f_flags;
                mode_t f_mode;
                loff_t f_pos;
                unsigned long f_reada, f_ramax, f_raend, f_ralen, f_rawin;
                struct fown_struct f_owner;
                int f_error;
                unsigned long f_version;
                void *private_data;};
```

Within this structure, the significance of data fields is as follows:

a) f_list is intended to link the file structure of one of the following lists.
   - sb→s_files (is the list that contains all the open files in the file system corresponding to that list), if the corresponding inod is not anonymous;
   - fs/file_table.c:free_list, containing all unused file type structures;
   - fs/file_table.c:anon_list, used only when is creating a new structure by get_empty_filp().

b) f_dentry corresponds to the director where the file which is attached;
c) f_vfsmnt is a pointer to the structure vfsmount of the file system that owns the file;
d) f_op represents a pointer to the file_operations structure;
e) f_count is a reference counter that is coordinated by get_file/put_filep/fput;
f) f_flags is use to hold flags for file;
g) f_mode is a combination of flags and modes for user space;
h) f_pos represents the current position of the pointer of read/write, stored on 64-bit;
i) f_reada, f_ramax, f_raend, f_ralen, f_rawin are fields used relatively to the operations of reading in advance;
j) f_owner provides information on the owner who will receive notifications of input/output SIGIO mechanism;
k) f_uid, f_gid refers to the file ID and group ID that belongs to that file, if that structure was created with get_emptz_filp ();
l) f_error allows the customer of NFS to return the warning messages concerning errors about editing;
m) f_version is used to invalidate cached addresses and is automatically incremented when changes f_pos;
n) private_data is used to hold private data on that file, which will be subsequently used by file systems.

Management of superblocks and mounting points

As shown, the Linux operating system can work with multiple file types. Thus, the corresponding information of file systems used by Linux at a time will be stored in two separate structures: super_block and vfsmount [NICOLAE, 2004, http://devlinux.org/namesys].

When analysing the parameters which was presented in this article, we should mention the fact that the Linux operating system allows to activate the same type of file system into multiple mount points, which means that the same structure from super_block can correspond to several vfsmount type structures [MUŞATESCU, 1999, NICOLAE, 2004, TANENBAUM, 1999].

The super_block structure type that is declared in the file include/linux/fs.h, must have the following:

```
struct super_block{
struct list_head s_list;
kdev_t s_dev;
unsigned long s_blocksize;
unsigned char s_blocksize_bits;
unsigned char s_lock;
unsigned char s_dirt;
struct file_system_type *s_type;
struct super_operations *s_op;
struct dquot_operations *dq_op;
unsigned long s_flags;unsigned long s_magic;
struct dentry *s_root;
wait_queue_head_t s_wait;
struct list_head s_dirty;
struct liste_hesd s_files;
struct block_device *s_bdev;
/* vfsmount belonging to this block */
struct list_head s_mounts;
/* diskquota specific options */
struct quota_mount_options s_dquot;
union{
struct minix_sb_info minix_sb;
struct ext2_sb_info ext2_sb;
/* all file systems that require private information belonging to the super_block */
void *generic?sbp;
}u;
/* the following field is only for virtual file system (VFS) */
struct semaphore s_vfs_rename_sem;
struct semaphore s_nfsd_free_path_sem;};
```

Within this structure of super_block, data fields used have the following meaning:

a) s_list indicates a double linked list that has active all of the super_blocks;
b) s_dev can have several meanings. Thus, for file systems that need a separate block where to be installed, this field represents the device's i_dev oriented blocks.
c) In the case of other types of file systems (called anonymous file systems), this field represents an integer returned by the function mkdev(unnamed_major, i), where „i" is the first bit not set in the table unnamed_dev_in_use and can have a value between 1 and 255;
d) s_blocksize represents the block size;
e) s_dirt will be set when the super_block will be modified and will be reset when they upgrade the memory support;
f) s_type is a pointer to the structure of type file of file system used at that time;
g) s_op is a pointer to the super_operations structure that contains the specific methods of work of that file system;
h) dq_op, s_dquot are associated fields when working with diskquota;
i) s_flags is to retain super_block's flags;
j) s_root allows to store some basic information about the file system used;
k) s_wait allows you to store files that are placed in the queue for the release of that super_bock;
l) s_dirty refers to the list of nodes modified;
m) s_files points to the list containing the names of all files that are open in that super_block;
n) s_bdev pointing to the block_device structure that stores the device where is installed that type of file system;
o) s_mounts points to a list of vfsmount type structures, one for each mounted instance of that super_block.

Discussion

Because of the way of implementation unit, by using similar attributes (types of lists, active, inactive pages or changed, etc.), the subsystem for memory management and the subsystem for management of files interacts very well without any conflicts during the operations desired by the users of the Linux operating system.

The structure indicated by the pointer type task_struct→files can be shared between a process "father" and a process "son" when the "son" was created by using the command clone() with the CLONE_FILES flag set.

When is open a new file, the allocated structure by field file_struct will be installed

in the location specified by the pointer current→files→[fd] fd.

Methods for working with files are contained in the file_operations structure, that is copied from the inode→i_fop and is declared in the file include/linx/fs.h.

We can say that this structure has a counterpart (vm_operations_struct) in the subsystem for management of the memory of a computer.

As a result of analysis performed on the structure and characteristics of the parameters of super_block, we can mention here that all the operations can be performed on a particular super_block who being mounted, are described very precisely by super_operations structure.

Conclusions

By the way of implementation, the Linux operating system allows users to work with different types of files, which represents a real advantage. It can be said that in working with file systems there are many similarities with the operations performed for the management of the memory of a computer.

These modern concepts presented are basic in design of current operating systems, in order to increase the performance concerning the speed of execution of operations and to minimize the space occupied by the storage media.

For this purpose we can use the techniques of pre-allocation of memory, delayed allocation techniques, using complex data structures such as simple lists, double lists, or the type of trees etc.

The concept of virtual is not used only for file management by the Linux operating system, it being used and at sockets, at working with the input/output devices, etc.

From those presented in this article it can be concluded that there is no a certain type of file system that can be catalogued as the best on all operating systems implemented in time or in relation to all types of processes or applications that are working with a computer, each one having specific features better performance or less, depending on the purpose of use.

References

1. Mușatescu Carmen, *Sisteme de operare*, Editura Radical **1999**
2. Nicolae Ileana, *Comparative Study of Memory Management Aspects al Linux and Windows*, Analele Universității din Craiova, **2004**
3. Tanenbaum, *Modern Concepts in Operating System Design*, Prentice Hal, **1999**
4. Vasile Cristian, *Optimization of Work Performance with Linux Operating System*, Banat Journal of Biotechnology, **2010**
5. ***, http://devlinux.org/namesys
6. ***, ftp://ftp.uk.linux.org/pub/linux
7. ***, www.linux.org/info/

Received: October 12, 2011
Accepted: November 16, 2011