DISTRIBUTED FILE SYSTEM

Iosif Irimie
(iiiosif@gmail.com)

Larisa Muntean
(larisa87_2008@yahoo.com)

Andrada Anca
(nightfixed@gmail.com)

Information Technology, I

Petru Maior University
Targu Mures

- January, 2010 -

Table of contents

Chapter I: Introduction

I.1. Purpose of this paper

The purpose of this paper is, primarily, an educational one. We aimed to test our abilities of understanding, finding solutions and solving problems by means of software programs and technology that has been made available to us.
In other words, learn to efficiently adapt, configure and use certain programs ment to perform specific actions.

I.2. Overview

Our team has been assigned with the task of designing, elaborating and implementing a working distributed file system, between workstations running on two different types of operating systems: Windows based and Unix based.
We had no restrictions related to how many work stations need to be linked and included into the system. We also had the liberty of choosing the network protocol that will be used. However, we've been asked to follow the main rule: maintain transparency.

Here is what we came up with: a LAN scaled system, including 4 workstations, using SMB as a network protocol. More details about our work are included in the following pages

I.3. The structure of the paper

The paper is divided into 4 chapters, and at the end it will present the conclusions of the team.
Chapter 2 contains a brief definition of the "file system" concept, as well as listing some of the differences between file systems used by the 2 types of operating systems, Windows based and Unix based. It will then introduce the concept of "distributed file systems" as well as point out some of its aspects (transparency, performance, concurrent file updates). It also includes references to possible choices of network protocols used in implementing a DFS.
Chapter 3 will present the scenario used in developing our distributed file system, as well as details about our task and our way of solving it.
Chapter 4 is reserved for conclusions.

Chapter II: File Systems

II.1 Definition

In a computer, a file system (sometimes written filesystem) is the way in which files are named and where they are placed logically for storage and retrieval.
The DOS, Windows, OS/2, Macintosh, and UNIX-based operating systems all have file systems in which files are placed somewhere in a hierarchical (tree) structure. A file is placed in a directory (folder in Windows) or subdirectory at the desired place in the tree structure.

File systems specify conventions for naming files. These conventions include the maximum number of characters in a name, which characters can be used, and, in some systems, how long the file name suffix can be. A file system also includes a format for specifying the path to a file through the structure of directories.

In our particular case, the term refers to the part of an operating system or an added-on program that supports a file system as defined in above. Examples of such add-on file systems include the Network File System (NFS).

II.2. File systems under Unix-like operating systems

Unix-like operating systems create a virtual file system, which makes all the files on all the devices appear to exist in a single hierarchy. This means, in those systems, there is one root directory, and every file existing on the system is located under it somewhere. Unix-like systems can use a RAM disk or network shared resource as its root directory

Unix-like systems assign a device name to each device, but this is not how the files on that device are accessed. Instead, to gain access to files on another device, the operating system must first be informed where in the directory tree those files should appear. This process is called mounting a file system. For example, to access the files on a CD-ROM, one must tell the operating system "Take the file system from this CD-ROM and make it appear under such-and-such directory". The directory given to the operating system is called the mount point - it might, for example, be /media. The /media directory exists on many Unix systems and is intended specifically for use as a mount point for removable media such as CDs, DVDs, USB drives or floppy disks. It may be empty, or it may contain subdirectories for mounting individual devices. Generally, only the administrator (i.e. root user) may authorize the mounting of file systems.
Unix-like operating systems often include software and tools that assist in the mounting process and provide it new functionality. Some of these strategies have been coined "auto-mounting" as a reflection of their purpose.

In many situations, file systems other than the root need to be available as soon as the operating system has booted. All Unix-like systems therefore provide a facility for mounting file systems at boot time. System administrators define these file systems in the configuration file (fstab in our case) which also indicates options and mount points. But, more about editing the fstab file will be discussed later.

II.3. File systems under Microsoft Windows

The File Allocation Table (FAT) filing system, supported by all versions of Microsoft Windows, was an evolution of that used in Microsoft's earlier operating system (MS-DOS which in turn was based on 86-DOS). FAT ultimately traces its roots back to the short-lived M-DOS project and Standalone disk BASIC before it. Over the years various features have been added to it, inspired by similar features found on file systems used by operating systems such as Unix.
Older versions of the FAT file system (FAT12 and FAT16) had file name length limits, a limit on the number of entries in the root directory of the file system and had restrictions on the maximum size of FAT-formatted disks or partitions. Specifically, FAT12 and FAT16 had a limit of 8 characters for the file name, and 3 characters for the extension (such as .exe). This is commonly referred to as the 8.3 filename limit.
FAT32 also addressed many of the limits in FAT12 and FAT16, but remains limited compared to NTFS

NTFS, introduced with the Windows NT operating system, allowed ACL-based permission control. Hard links, multiple file streams, attribute indexing, quota tracking, sparse files, encryption, compression, reparse points (directories working as mount-points for other file systems, symlinks, junctions, remote storage links) are also supported, though not all these features are well-documented.

Unlike many other operating systems, Windows uses a drive letter abstraction at the user level to distinguish one disk or partition from another. For example, the path C:\WINDOWS represents a directory WINDOWS on the partition represented by the letter C. The C drive is most commonly used for the primary hard disk partition, on which Windows is usually installed and from which it boots. This "tradition" has become so firmly ingrained that bugs came about in older versions of Windows which made assumptions that the drive that the operating system was installed on was C. The tradition of using "C" for the drive letter can be traced to MS-DOS, where the letters A and B were reserved for up to two floppy disk drives. Network drives may also be mapped to drive letters

II.4. Distributed file system

A distributed file system or network file system is any file system that allows access to files from multiple hosts sharing via a computer network. This makes it possible for multiple users on multiple machines to share files and storage resources. The client nodes do not

have direct access to the underlying block storage but interact over the network using a protocol. This makes it possible to restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.

Transparency – this is usually built into distributed file systems, so that files accessed over the network can be treated the same as files on local disk by programs and users. The multiplicity and dispersion of servers and storage devices are thus made invisible. It is up to the network file system to locate the files and to arrange for the transport of the data.

Performance - in a network file system, a remote access has additional overhead due to the distributed structure. This includes the time to deliver the request to a server, the time to deliver the response to the client, and for each direction, a CPU overhead of running the communication protocol software. The performance of a network file system can be viewed as one dimension of its transparency.

Concurrency control becomes an issue when more than one person or client are accessing the same files and want to update it. Hence updates to the file from one client should not interfere with access and updates from other clients. Concurrency control or locking may be either built into the file system or be provided by an add-on protocol.

II.5. Network protocols used in implementing a DFS

Common Internet File System (CIFS) is a protocol that lets programs make requests for files and services on remote computers on the Internet. CIFS uses the client/server programming model. A client program makes a request of a server program (usually in another computer) for access to a file or to pass a message to a program that runs in the server computer. The server takes the requested action and returns a response.

CIFS is a public or open variation of the Server Message Block Protocol developed and used by Microsoft. Like the SMB protocol, CIFS runs at a higher level than and uses the Internet's TCP/IP protocol. CIFS is viewed as a complement to the existing Internet application protocols such as the File Transfer Protocol (FTP) and the Hypertext Transfer Protocol (HTTP).

CIFS lets you…

- Get access to files that are local to the server and read and write to them
- Share files with other clients using special locks
- Restore connections automatically in case of network failure
- Use Unicode file names

The Network File System (NFS) is a client/server application that lets a computer user view and optionally store and update file on a remote computer as though they were on the

user's own computer. The user's system needs to have an NFS client and the other computer needs the NFS server. Both of them require that you also have TCP/IP installed since the NFS server and client use TCP/IP as the program that sends the files and updates back and forth. NFS was developed by Sun Microsystems and has been designated a file server standard. Its **protocol** uses the Remote Procedure Call (RPC) method of communication between computers.

Direct Access File System (DAFS) is a network file system, similar to Network File System (NFS) and Common Internet File System (CIFS), that allows applications to transfer data while bypassing operating system control, buffering, and network protocol operations that can bottleneck throughput. DAFS uses the Virtual Interface (VI) architecture as its underlying transport mechanism. Using VI hardware, an application transfers data to and from application buffers without using the operating system, which frees up the processor and operating system for other processes and allows files to be accessed by servers using several different operating systems. DAFS is designed and optimized for clustered, shared-file network environments that are commonly used for Internet, e-commerce, and database applications. DAFS is optimized for high-bandwidth networks, and it works with any interconnection that supports VI.

Other types of client-server applications and protocols might be used in implementing a distributed file system, such as NCP, AFS, CODA. However, these were designed for high scalability, and they do not fit in the discussion that this paper approaches**.**

Chapter III. Assignment Scenario

III.1.Task

Design, develop and implement a distributed file system, making use of the available software applications and technology.

III.2.Choosing the workstations, server and communications protocol

For this project we picked 4 workstation (desktop PCs) and 1 server.

The desktop PCs, later referred to as workstations, are operating on Windows XP. The server is operating a debian-based Linux distribution, Ubuntu 9.10. SMB protocol will be used for communications. SAMBA server will be installed and configured on the server.

Why Windows XP on the workstations?
For no specific reason, other than the simple one: it was the only option available. Apart from this, it's easier to avoid any compatibility issues.

Why Ubuntu 9.10 on the server?
Firstly, because it's user friendly and very easy to configure and set up. Secondly, because every one of us (the members of the team) have worked with Ubuntu (different releases however) before and were familiar with the commands and structure.

Why go for SAMBA?
Here there might be several reasons, and it's arguable if we made the best choice here. An alternative would have been NFS.

The difference between Samba and NFS is primarily that Samba uses the SMB protocol which is considered "standard" for PCs. Windows and OS/2 both have built in support for it, a free client is also available for DOS, I'm not sure about MacOS), whereas NFS uses its own protocol (usually just called "NFS") which is not commonly available for PCs (NFS clients do exist for operating systems other than UNIX/Linux, but they're usually neither free or easy to setup). Samba supports mounting of CIFS file systems.

Samba's SMB protocol allows the server machine to handle authentication, so it can decide what files the client has access to based on the particular machine and user connecting. NFS by default trusts all client machines completely (it's really not intended to share files to unsecured workstations) and lets the client machines handle authentication all on their own (once an NFS server has been told to accept connections from a client machine through its configuration file, the client does not require any further server-side authentication, and can do anything it wants with the filesystem NFS gives it access to).

A Samba server offers the following services:

- share one or more directory trees
- share one or more distributed file system
- share printers installed on the server, among Windows clients on the network
- assist clients with network browsing
- authenticate clients logging onto a Windows domain
- provide or assist with Windows Internet Name Service (WINS) name – server resolution

Asume that we have the following basic configuration: the server, operating Ubuntu 9.10 with samba installed on it, code named “Sun”, with the following workstations: Jupiter, Mars, Mercury and Saturn. These are all connected through a local area network (LAN).

It is mandatory for all the computers to share the same workgroup: Milky Way.
A workgroup is a group name tag that identifies an arbitrary collection of computers and their resources on an SMB network. Several workgroups can be on the network at any time, but for our basic network example, we'll have only one.

It must be mentioned, that this paper focuses mainly on configuring the Samba server. Details about how the workstations configuration was done will be briefly given later.

III.3. The Samba server “Sun”

Before getting to the point of installing and configuring samba, a few notes must be taken.

As we said above, Samba's SMB protocol allows the server machine to handle authentication, so it can decide what files the client has access to based on the particular machine and user connecting. This means that if we want a user to have access to the samba shares, we must first give that user access to Sun. For this purpose, 2 new unprivileged user accounts will be added on Sun: Jupiter and Saturn. The assigned passwords and/or home directories are not important, as samba will need different passwords.

With that being said and done, let’s move on: **installing the samba package.**

To do this on a debian-based Linux distribution such as Ubuntu, we used the simple command (logged in as a normal user, therefore borrowing the root powers):

sudo apt get install samba

Samba services are implemented as two daemons:
- smbd, which provides the file and printer sharing services
- nmbd, which provides the NetBIOS-to-IP-address name service

In Unix and other computer multitasking operating systems, a **daemon** is a computer program that runs in the background, rather than under the direct control of a user; they are usually initiated as background processes.

Samba configuration is achieved by editing a single file (in our case, located at /etc/samba/smb.conf).

A simple SMB connection is achieved in 3 steps:

1. Establish a NetBIOS session
2. Negotiate the protocol variant
3. Set session parameters and make a tree connection to the resource

The Samba configuration file called *smb.conf* in our case, uses the same format as Windows *.ini* files. Its format is simple and easy to learn, and every of its sections contains a large amount of text providing explanations and comments related to every option that can be configured. To mention that all the .conf files on the Sun server will be edited using the text editor tool "gedit".

For our project, we've configured Sun to act in the following way:

- a samba share, called "collections" is created on Sun; after that, every user is given the permission to access, read, write in that folder

sudo mkdir /srv/samba/collections
sudo chown nobody.nogroup /srv/samba/collections
sudo chmod 667 /srv/samba/collections

The first line created the folder; the second one sets the user permissions, so basically the folder has no owner; the third line grants read/write/execute permissions for everybody.

In our example, we used ***/srv/samba/collections*** as a path for our samba share, because according to the *Filesystem Hierarchy Standard (FHS),* /srv is where site-specific data should be served. Technically Samba shares can be placed anywhere on the filesystem as long as the permissions are correct, but adhering to standards is recommended.

- "Collections" is then used as a mount point for different collected files and folders from Mars and Mercury
- later on, "Collections" is shared by Sun, with Jupiter and Saturn.

In other words, Sun picks up the files/folders from Mars and Mercury and places them into Collections; when connecting to access Collections, Jupiter and Saturn will see it as a single folder, without seeing where do the files inside it come from, therefore the system maintains transparency.

To achieve this, here is how we edited the ***smb.conf*** configuration file:

1. Under the ***Global Settings*** section,

workgroup = Milky Way //changed according to what we mentioned earlier
netbios name = Sun //changed to reflect the name of the server
level2 oplocks = True // the importance of this setting will be discussed later

2. Under the ***Authentication*** section,

security = user //changed to remove the comment; this will imply that every user trying to access the samba shares must first have access on Sun

smb passwd file = /etc/smbpasswd //changed so the option **smb passwd** file specifies the path to the encrypted smbpasswd file. The smbpasswd file is a copy of the /etc/passwd file of the Ubuntu system containing valid usernames and passwords of clients allowed to connect to the Samba server. The Samba software reads this file, smbpasswd when a connection is requested.

3. Under the ***Share definitions*** section,

[Collections]
comment = a collection of files to be shared
path = /media/collections //the path for the share
browseable = yes //enables Windows clients to browse the shared directory using Windows Explorer
guest ok = no //if changed to “yes”, it would allow users to connect without password
read only = no //gives write access to the shared directory

Now that samba is set up and we have also told her what folder to share over to Jupiter and Saturn.

With all this being said and done, it’s time to restart the samba daemons:

sudo /etc/init.d/samba restart

So far, we have managed to setup samba server on Sun, configure it, create and specify the share “Collections” and instruct samba which users will be allowed to access the share.

It is time to move on, to put stuff inside Collections, which right now, stands empty.

We have talked before about **mounting** files under Unix based operating systems.

In order to put items inside out share called “Collections” we are going to have to mount them, therefore “Collections” becoming the mount point.
In order to keep things organized, inside the “collections” folder, we have created specific folders for each data type (for instance, “documents” folder for documents, “music” and

"pictures" respectively). The beauty of Ubuntu is that all the folders created inside the "collections" folder will inherit its attributes and properties.

Mounting a file from a Windows workstation, in our case, Mars and Mercury, can be done in 2 steps. All Unix-based operating systems, and in our case Ubuntu, are using a configuration file called "**fstab" (short for or *file systems table*) in conjunction with the mount operation.**
"fstab" is a configuration file that contains information of all the partitions and storage devices on the computer. The file is located under /etc, so the full path to this file is /etc/fstab.

In order for our server "Sun" to mount the folders, we first had to edit the fstab file. Every line added to the fstab file has the following format:

<device> <mountpoint> <filesystemtype> <options> <dump> <pass>

In our case, this would translate into…

//mars/music /srv/samba/collections/music cifs exec, 0 0
//mars/documents /srv/samba/collections/documents cifs exec, 0 0

- //mars/music – specifies the location of the folder that we're mounting; "mars" is the windows workstation, the shared folder is called "music"
- then we specify the location where the folder will be mounted
- then we specify the file system type: CIFS
- exec lets you execute binaries that are on that partition, and it's the default setting (noexec is the alternative)
- the first 0 is the "dump" value, which determines if the filesystem should be backed up or not; "0" means "ignore it".
- if the last option is a 0 again, it means the filesystem doesn't need a fsck run

If the workstation from which we are mounting the files would require a username and password to authenticate, then the above lines would look like this (for example):

//mars/music /srv/samba/collections/music cifs exec,username=username, password=password 0 0

Normally, Ubuntu mounts the files specified inside /etc/fstab every time it boots up. If the /etc/fstab file was edited with Ubuntu running and without rebooting for the changes to take effect, then the files need to be mounted manually: (with them being mounted automatically at the next reboot ofcourse)

III.4.Configuring the windows workstations

We're going to talk here about configuring these 4 windows stations.
There are 2 configurations sets: 1 set for the windows stations that will share folders FOR Sun and 1 set for the windows stations that ACCESS Sun and read information. No need to specify again that every one of them needs to be part of the MilkyWay workgroup.

Configuring the Windows stations is much easier than everything that has been done so far:

On each of the 2 PCs, Mars and Mercury, we need to make sure that the files that will be mounted on Sun have the "share" property set.

On each of the 2 PCs Jupiter and Saturn respectively, a new network place will be added. The format is as a below:

\\Sun's IP address\\srv\samba\collections

Upon trying to connect to the "Collections" share, every user will be asked for their respective username and password. After typing them in correctly, they will be given access to all the contents inside "Collections".
Logging in to access the shares is only required once, hence the first time the share is accessed and added into the "network locations" section of the windows machine.

When we talked about configuring the **smb.conf** file on Sun, under the "global" section for settings, we've inserted a line that we didn't comment:

level2 oplocks = True

- a lock is set on the file, but with discrimination on read/write operations; for example, making the file available in read only mode.

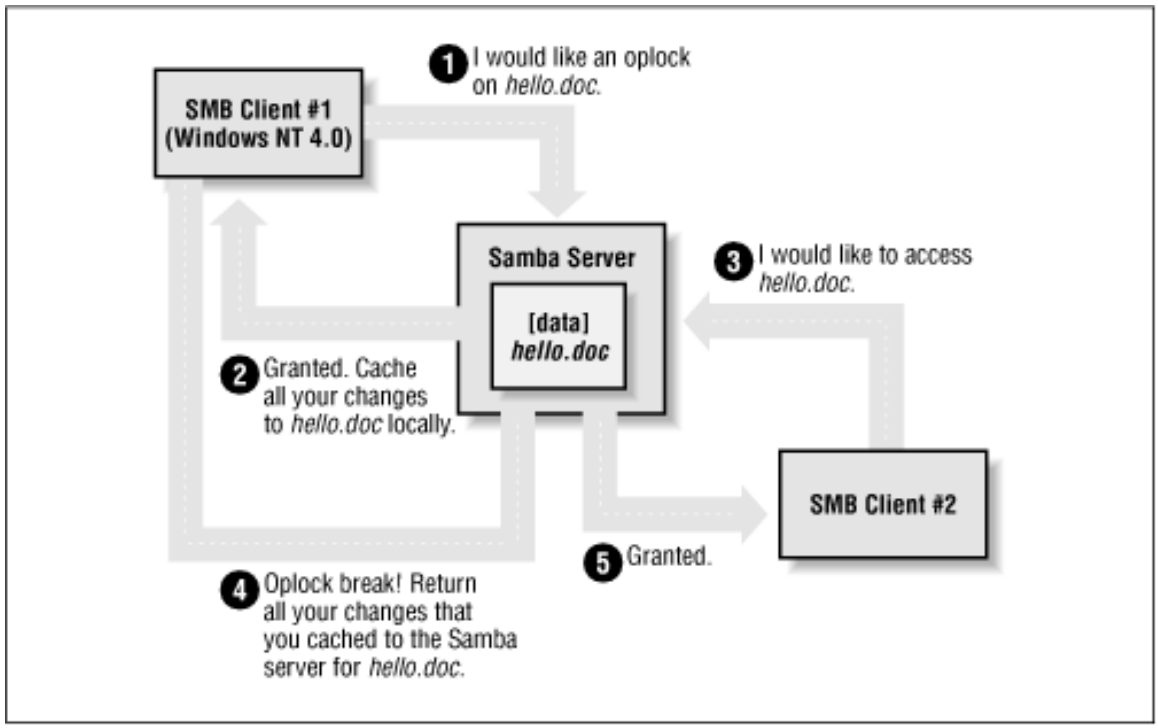
It's time to say a few words about another very important aspect of a distributed file system: file system consistency.

In a distributed system file environment, it is very common that multiple users may access the same resource (in our case, a file) at the same time. Concurrent writes to a single file are not desirable in any operating system. To prevent this, most operating systems use locks to guarantee that only one process can write to a file at a time. Operating systems traditionally lock entire files, although newer ones allow a range of bytes within a file to be locked. If another process attempts to write to a file (or section of one) that is already locked, it will receive an error from the operating system and will wait until the lock is released.

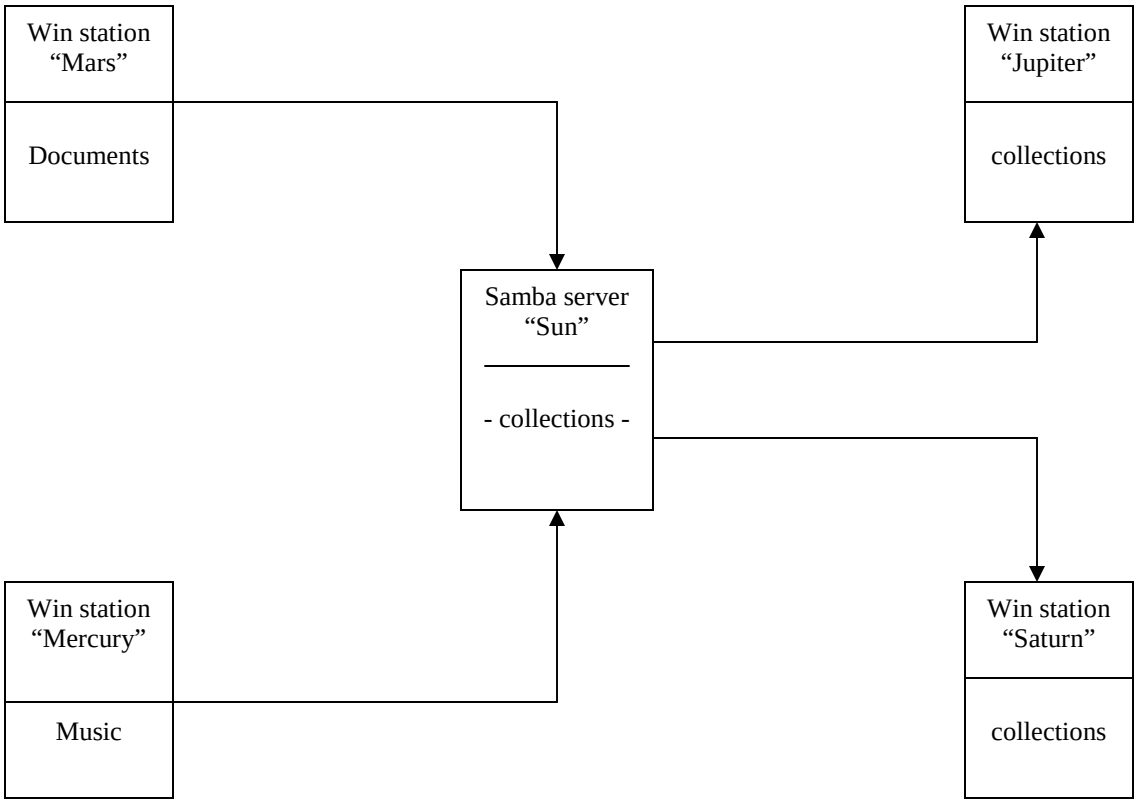
Samba supports the standard DOS and NT file system (deny-mode) locking requests, which allow only one process to write to an entire file on a server at a give time, as well as byte-

range locking. In addition, Samba supports a new locking mechanism known in the Windows NT world as opportunistic locking - oplock for short.

Opportunistic locking allows a client to notify the Samba server that it will not only be the exclusive writer of a file, but will also cache its changes to that file on its own machine (and not on the Samba server) in order to speed up file access for that client. When Samba knows that a file has been opportunistically locked by a client, it marks its version as having an opportunistic lock and waits for the client to complete work on the file, at which point it expects the client to send the final changes back to the Samba server for synchronization. If a second client requests access to that file before the first client has finished working on it, Samba can send an oplock break request to the first client. This tells the client to stop caching its changes and return the current state of the file to the server so that the interrupting client can use it as it sees fit. The way in which Samba works with locks can be seen in the image below: (image taken from http://oreilly.com/catalog/samba/chapter/book/ch05_05.html)



We've also attached below a diagram, to get the idea of how our system works (finally):



Explanation:

The 2 windows stations Mars and Mercury are sharing folders for the Server Sun, which compiles them in the "Collections" share and offers them for accessing towards the other 2 windows stations Saturn and Jupiter.

Chapter IV: Conclusions

With this project, we have managed to get a glimpse of how complicated and complex the world of a distributed system (be it file system, operating system) is. We are open to suggestions and forthcoming discussions. But before that, some observations must be made.

Firstly, regarding security. This is a topic that we haven't talked much about, except for when we decided that authentication for the samba share should be done on the server.

Secondly, about the network protocol used. We have decided to go for Samba alone; it would have been a good idea to try and connect at least 1 of the workstarions through NFS. But then again, concurrent file access problems might have occurred;
Problems can occurr if the same file is accessed by both SMB and NFS at the same time. You can however prevent these problems from happening through careful server configuration (allowing NFS and Samba to share the same file system without any problems). This is one area which we haven't experienced yet.

Thirdly, we think that we should have focused more on the DSF consistency aspect. We did work with setting lock permitions through Samba but we think that we could have done more in regarding the testing of each setup option.

We've picked a system that has very low scalability: such a design, can only be implemented over a local area network. Another downside of this is that we have no file caching or replication mechanism. This means that if the workstations from which the data is mounted (Mars and Mercury respectively) would happen to go offline or disconnect, the shares on Sun would become inaccessible.

What we can say, is that we've only managed to scratch the surface of the huge shell that conceals the secrets of implementing a succesuful distributed file system.

References:

- http://whatis.techtarget.com/ - for specific and precise terms definitions in English
- http://en.wikipedia.org/ - for specific terms definitions
- http://ubuntuforums.org/ and specifically http://ubuntuforums.org/showthread.php?t=202605 for detailed instructions on how to setup the samba server
- http://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/ for detailed information on how samba and its daemons work
- http://oreilly.com/catalog/samba/chapter/book/ a more in-depth guide with examples of everything that samba can do for you
- http://www.tuxfiles.org/ - for specific explanations and detailed examples of configuration files editing
- http://www.faqs.org/docs/securing/chap29sec284.html for a detailed example of a smb.conf edited file
- http://searchenterpriselinux.techtarget.com/news/article/0,289142,sid39_gci921657,00.html for information on how networked file systems work
- http://lists.samba.org/archive/samba/2000-June/019414.html for brief explanation on how samba understands network protocols
- http://www.ubuntugeek.com – for tips and tricks on mounting windows shares