

Lab 8

File Systems Management

ITSC205: Operating Systems Internals

NAME:Ahmed Almass

Table of Contents

[Lab Outcome(s) 3](#_Toc510478789)

[Reading 3](#_Toc510478790)

[Introduction 3](#_Toc510478791)

[1.0 NTFS File Types 4](#_Toc510478792)

[2.0 Windows and Linux File System Types 5](#_Toc510478793)

[3.0 NTFS Internals 5](#_Toc510478794)

[4.0 Linux File System Analysis 8](#_Toc510478796)

[5.0 File System Calls and Buffer Size Analysis 13](#_Toc510478796)

|  |  |  |
| --- | --- | --- |
| NTFS File Types | 5 |  |
| Windows and Linux File System Types | 5 |  |
| NTFS Internals | 10 |  |
| Linux File Systems Analysis | 10 |  |
| Ext File System Journaling feature | 5 |  |
| Ext File System hard and symbolic links feature | 5 |  |
| File System calls and Buffer sizes analysis | 10 |  |
| TOTAL MARK | 50 |  |

Lab Outcome(s)

* Examine file types and file system types.
* Analyze the NTFS structure.
* Analyze Linux ext file system features

Reading

* Textbook chapter 13.- File system Interface and Chapter 14 File System Implementation

Introduction

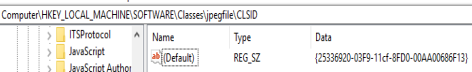
Ext4 is the default file system for most Linux distributions and has features such as: ability to use file systems larger than 16TB, nsec time stamps, extent format reduces metadata overhead, journal checksum, persistent file pre-allocation (e.g for streaming media, databases) and it supports large block size.

Windows NTFS file system has become very popular due to its reasonable efficiency, log-based journaling, encryption support, compression support, and simply because it is the primary file system of the most popular (by number of installed copies) operating system in the world. Windows also supports other file systems to maintain compatibility with older systems (e.g., FAT compatibility)

In this lab Windows and Linux utilities will be used to explore, monitor and debug file systems features.

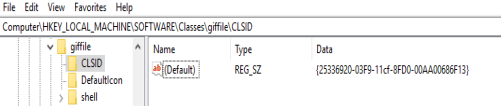
1. NTFS File Types \_\_\_\_/5

NTFS supports a wide range of file types by associating a default program with each type of file. Whenever a file is opened, that default program is run to process that file. File types are identified by the extension portion of the file name (e.g., .doc).

1. Windows 10 hides the file extension of known files. In order to display file extensions, do the following:
   1. Click on Start menu  type or run File Explorer Options command.
   2. Explore View tab  Advanced Settings and uncheck “Hide extensions for known file types” Open File Explorer and go to C:\Windows folder to verify file extensions.
   3. Check the type column and identify three file types based on file extension.
2. Application
3. Bin File
4. DAT
5. The registry key that defines (among other things) file type associations is HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes. Find the subkey for .jpg files. The value PersistentHandler is a Globally Unique Identifier of a code object that handles that file type. What are the last five characters of the data of this value?

6IF13

1. Compare this data to that of the subkey for .gif files. What conclusion can you draw from this?



86F13

“A globally unique identifier (GUID) is a 128-bit number created by the Windows operating system or another Windows application to uniquely identify specific components, hardware, software, files, user accounts, database entries and other items.

1. Windows and Linux File Systems Types \_\_\_/5
2. Insert USB or SD card and Use Disk management tool to identify current mounted file systems for Windows and Linux operating systems. Complete the following table:

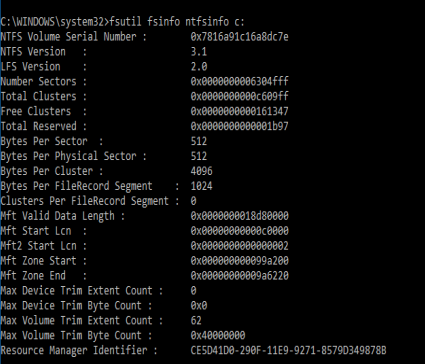
|  |  |  |
| --- | --- | --- |
| **Device Name** | **File System Type** | **Volume Size** |
| **Windows**  Hard disk C:\  Volumes on the VM and their respective file systems: | NTFS | 49.51 |
| **Linux**  Volumes and their respective file systems: | ext4 | 454 |
| **Windows**  USB or SD card |  |  |
| **Linux**  USB or SD card |  |  |

1. NTFS Internals \_\_\_/10

There are different windows and system internals utilities that can be used to examine and configure NTFS settings.

**Windows Fsutil utility**

1. Open cmd as administrator, type and explore fsutil Windows utility to examine NTFS file system features. Use fsutil fsinfo ntfsinfo c: to query NTFS specific volume information and answer the following questions



1. What is the purpose of MFT?

All information about a file, including its size, time and date stamps, permissions, and data content, is stored either in MFT entries, or in space outside the MFT that is described by MFT entries

1. Number of bytes per physical sector:

512

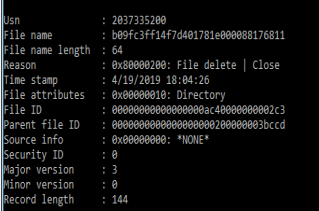
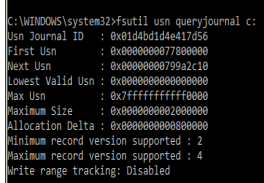
1. Cluster size:

4096

1. Every entry in MFT table is a record: What are the bytes per FileRecord segment?

1024

1. The free space is shown in hexadecimal format. Convert this to decimal using the Windows Calculator in programmer mode. From the result, calculate the free space in bytes.
2. You can use the fsutil to create, delete, or query journal information. Explore fsutil usn queryjournal c: and fsutil usn readjournal c: | more and answer the following questions:
   * 1. What is the purpose of USN Journal file?



(USN) Updated Sequence Number Journal

maintains a record of changes made to the volume.

* + 1. How can you use this information in security?

Its good for forensics point of view to identify suspesous iles

NTFS Hard and Soft Links

There are two ways you can create a hard link: the *fsutil hardlink create* command or the *mklink* utility with the /*H* option. In this experiment we'll use *mklink* because we'll use this utility later to create a symbolic link as well.

1. Open cmd as Administrator and create a file called test.txt and add some text to it, as follows:

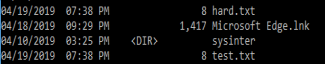
C:\>echo hello > test.txt

1. Now create a hard link called hard.txt using mklink command

C:\>mklink hard.txt test.txt /H

You should see - Hardlink created for hard.txt <<===>> test.txt

1. Type dir and compare creation time and files sizes. Are the file sizes different? Why?



Another file with a link to the same underlying inode.

Same creation

1. Now create a soft link for the test.txt file as follows:

C:\>mklink soft.txt test.txt

You should see - symbolic link created for soft.txt <<===>> test.txt

1. Type dir and compare files creation time and sizes. What is different?



Doesnt say the size of soft.txt but different time of creation.

Special kind of file that points to another file.

1. What would happen if you delete the original test.txt file? Analyze hard link and soft link files and explain differences to instructor

Delete original file will remove soft.txt but not the hard.txt.

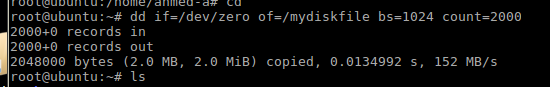
4.0 Linux File Systems Analysis \_\_\_/10

In order to practice Linux File Systems features we will create a virtual partition and create different **ext** file systems versions.

An ordinary file can be used as a virtual disk partition and act just like a disk. The Linux loopback virtual file system performs input and output on a file as if it is a filesystem. Before the file can be used to store directories and files, it must be formatted to hold one of the various Linux file system types.

1. Use **su** to change to root user
2. Use **cd** command to move to **/**
3. Create an empty 20MB file using **dd** command . Set the input file (if) to the /dev/zero device (a pseudo-device that simply outputs zeros) as follows:

dd if=/dev/zero of=/mydiskfile bs=1024 count=20000



* 1. Use **ls –l** command to verify file name created and its size.



* 1. Use Linux manual (man) to find the purpose of dd command and its usage. Study the command **dd** parameters to explain how the command generates a file of 20MB size:

dd: copy a file converting and formatting according to the operand.

If=/dev/zero : read from file instead of stdin

of=/mydiskfile: write to file instead of stdout

bs=1024 read and write up to 1024 bytes at a time

count=2000 copy only N input blocks

1. To use the file **like a filesystem**, associate it with a loopback device using **losetup** command. Use Linux manual to find the purpose of losetup command and its usage

losetup /dev/loop0 /mydiskfile

The file you have created now acts like a *block* I/O device (one that inputs and outputs blocks of characters like a disk drive, as opposed to a *character* I/O device that inputs and outputs single characters like a keyboard) called **/dev/loop0**.

1. Whenever a new device is mounted (connected) to a file system, the starting point in the directory structure must be specified. This is the *mount point*. For example, if the mount point of deviceA is /mnt/deviceA, then the root directory of all files on device is /mnt/deviceA/. Create an empty directory to act as a mount point for the loop0 device:

mkdir /mnt/loop0

1. **Create Ext File Systems and mount it on /dev/loop0**

A file system is a high level format applied to a block I/O device. Examine the properties of different file systems including **ext2, ext3,ext4**. The **mkfs** command is a front end for a number of related utilities that create specific file systems. Read Linux man to learn the usage of mkfs. Invoke mkfs by specifying a file system type on the command line. ext4 on newer Linux distributions.

1. Complete the following table information for each ext file system version and the respective block size.
2. **Create a file system** with the respective block size as follows: (use the file type and block size indicated in the table.

mkfs.ext2 –b <block\_size> /dev/loop0

**Replace** <block\_size> with the respective block size specified in the following table

To modify inode size use the option –i and to add journaling to ext2 use option -j

e.g mkfs.ext2 -b 1024 -i 4096 /dev/loop0

1. Analyze the results after creating each file system and record the the first two columns total inodes and total blocks used by each file system indicated in the table. Once the file system is created, use the mount command as follows to mount loop0 device. Then use df command to verify file system usage and now record the last column to specified how much was used by the file system
2. **Mount the file system** (type is ext2, ext3, etc.) as follows:

mount –t <type> /dev/loop0 /mnt/loop0

**Replace** <type> with the respective file system type

1. **Check the capacity (usage) of the file system** using **df** command and record it in the third column of the table.
2. You can also use df -i command to verify the inodes usage
3. **Unmount** each file system by using **umount /mnt/loop0** command and repeat the previous steps for **each raw** of the table.
4. **Do steps a  e for each entry in the table that follows.**

|  |  |  |  |
| --- | --- | --- | --- |
| **File System Details** | **Total inodes** | **Total Blocks** | **Use df command and record the “Used” column** |
| ext2 with block size 1024 and 4096 bytes/inode |  |  |  |
| ext2 with block size 2048 |  |  |  |
| ext2 with block size 4096 |  |  |  |
| ext2 with Journal File and block size 1024 |  |  |  |
| ext3 with block size 1024 |  |  |  |
| ext4 with block size 1024 |  |  |  |

1. Based on the information gathered in the previous question, which file system results in the lowest overhead and greatest useable capacity? Does that make it the best file system (why / why not)?
2. **Ext File System Journaling feature \_\_\_/5**

***Journaling*** allows rapid recovery from many file system errors by recording every file system transaction. Many modern file systems use this feature.

1. Use **mkfs** command to create an ext2 file system with 1024 byte blocks on device **/dev/loop0**
2. Use **mount** command **to** mount ext2 on /mnt/loop0.
3. Use **df** or just **mount** command to verify if file system ext2 was mounted
4. Use man to learn about tune2fs command. What is the purpose of this command?
5. Use **tune2fs** command with respective option to add journaling feature to ext2 file system as follows:

tune2fs -j /dev/loop0

1. Use cd to change and access mounted ext2 file system on /mnt/loop0

cd /mnt/loop0

1. Use **ls –ail** command to display the hidden journal file with the respective inode and record the size and name of the journal file:
2. **Demonstrate** to the instructor that the journal file was created.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

1. Use df command and analyze the file system usage. For comparison, create an ext3 file system with 1024 byte blocks and ext4 with 1024. Compare file systems usage with file system ext2 with journaling. What is your conclusion?
2. **Ext Hard and Symbolic Links features \_\_\_/5**
3. Create an ext4 file system on /dev/loop0 and mount it
4. Use the command ***df*** to verify if ext4 file system was mounted
5. Access the mounted file system by using **cd /mnt/loop0**
6. Create a directory called testdir on that device (mnt/loop0)
7. Use the command touch to create an empty file called testfile in the same device (/mnt/loop0)
8. Edit testfile file and type “Hello”
9. Use the command ***info ln*** and read section 12.2 to clarify and differentiate hard links and soft links.
10. Use the command **ln –s source target** to create a soft link for both the testdir directory and the testfile file
11. Use the command **ln source target** to create a hard link for both the testfile file and the testdir directory. Can you create hard link for directories? Why ?
12. Display **inodes** for the directory and file with **ls –ail**.
13. Demo to instructor inodes values for the soft link and hard link and compare it with the inode of the original file (testfile). Explain results.
14. Modify the content of testfile file and compare it with the content of the hard link file.

Explain to instructor:

* + 1. Why is the content of the hard link file same as the original file content?
    2. What happens if you delete the original file while it has a hard link to it?
    3. What happens if you delete the original file while it has a symbolic link to it?

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

5.0 File system calls and buffer size \_\_\_/10

1. Write down the following newcp.c code
2. Save the following C program in the home directory as newcp.c.

/\*\* newcp.c new version of cp - uses read and write with tunable buffer size

\* usage: cp1 src dest \*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define BUFFERSIZE 4096

#define COPYMODE 0644

void oops(char \*s1, char \*s2)

{

fprintf(stderr,"Error: %s ", s1);

perror(s2);

exit(1);

}

main(int ac, char \*av[])

{

int in\_fd, out\_fd, n\_chars;

char buf[BUFFERSIZE];

if ( ac != 3) {

fprintf( stderr, "usage: %s source destination\n", \*av);

exit(1;

}

/\* open files \*/

if ( (in\_fd=open(av[1], O\_RDONLY)) == -1 )

oops("Cannot open ", av[1]);

if ( (out\_fd=creat( av[2], COPYMODE)) == -1 )

oops( "Cannot creat", av[2]);

/\* copy files \*/

while ( (n\_chars = read(in\_fd , buf, BUFFERSIZE)) > 0 )

if ( write( out\_fd, buf, n\_chars ) != n\_chars )

oops("Write error to ", av[2]);

if ( n\_chars == -1 )

oops("Read error from ", av[1]);

/\* close files \*/

if ( close(in\_fd) == -1 || close(out\_fd) == -1 )

oops("Error closing files","");

}

1. Compile the program with gcc –o newcp newcp.c.

**Note** that newcp operates just like the cp file copy command.

1. Run the program using the following syntax:

**./newcp source destination** (Replace source and destination with the respective path of the source and destination files e.g ./newcp /mydiskfile junkfile

**Modify buffer size and analyze results**

1. Edit newcp.c program and modify the buffer size as suggested in the table below. Every time you make changes on the file you need to recompile the program
2. Run the newcp program using mydiskfile (20 MB) file as source and Junkfile as destination for each of the buffer size indicated in the table below. Use the following commands for each buffer size and record the results in the table.
   1. Use the **time** command to analyze user and system time for each buffer size **time ./newcp /mydiskfile junkfile**
   2. Use **strace –c ./newcp /mydiskfile** **junkfile** to identify the main system calls used in this program. Record only write() system calls required for different buffer sizes.

|  |  |  |
| --- | --- | --- |
| Buffer size | Execution time (user + system) in sec. | **Only write()** system calls |
| 4096 |  |  |
| 2048 |  |  |
| 1024 |  |  |
| 512 |  |  |
| 4 |  |  |

1. List the effects of decreasing the buffer size in this program.
2. Demonstrate to the instructor the results and explain what caused these effects.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |