

Lab 7

File Systems Management

ITSC205: Operating Systems Internals

NAME: \_**Coleton Sanheim**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 6](#_Toc510478794)

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

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

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies

|  |  |  |
| --- | --- | --- |
| NTFS File Types | 5 |  |
| Windows, Linux and other systems 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, the application associated with the file will run and process the file. Windows file types are identified by the extension portion of the file name (e.g., .doc).

1. Windows 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 and write down three different file types based on file extension.

**.exe > Application, .dat > DAT file, .xml > XML Document**

1. 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?

**0bfeb**

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

**They are both image type files, as they both end with 0bfeb**

1. Windows and Linux File Systems Types \_\_\_/5
2. Insert USB or SD card. Use Windows Disk management tool to identify current mounted file systems for Windows and use df or mount command to verify mounted file systems in 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** | **1862.41 GB** |
| **Linux**  /  Volumes and their respective file systems: | **ext4** | **30G** |
| **Windows and Linux**  USB or SD card | **FAT** | **1.87 GB** |

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:
2. What is the purpose of MFT?

**It contains metadata for all the NTFS volumes**

1. What is the number of bytes per physical sector ?

**512**

1. What is the Cluster size?

**4096 bytes**

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

**1024**

1. What is the free space in bytes?

**2,087,387,136 bytes**

1. 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?

**determine which files have changed since the last backup so that only files that have changed are added to the history so far**

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

**Use it as a checksum to determine if files have been maliciously changed**

NTFS Hard and Soft Links

There are two ways you can create a Windows 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 > originalfile.txt

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

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

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

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

**They are the same size and have the same creation time, because it is a hard link which is a direct copy of the original**

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

C:\>mklink soft.txt originalfile.txt

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

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

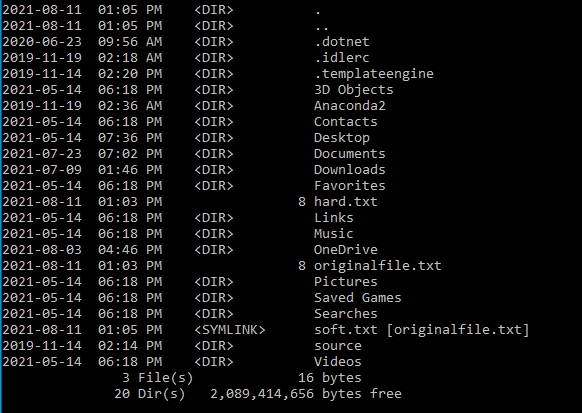
**A softlink is not recognised as a file but instead as a symlink and as such does not have a size associated with it, and the creation time is not the same.**

1. What would happen if you delete originalfile.txt file? Analyze hard link and soft link files. Attach screen captures that demo:
   1. Creation of hard and soft link for originalfile.txt

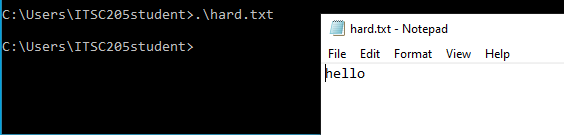




* 1. Files creation time and sizes



* 1. Deletion of originalfile.txt and results of hard and soft links





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

**If=FILE < reads from FILE, of=FILE < writes to FILE, bs=BYTES < read/write BYTES at a time, count=N < copy only N input blocks.**

**So it reads from /dev/zero to a new file and copys 20000 blocks of data from it**

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

NOTE: Make sure that ***loop0*** device is available. Use ***df*** command to verify if loop devices are mounted. If loop devices are mounted use the command: sudo umount /dev/loop\* use df command to verify if they were unmounted. If that does not work use loop device that is not in used or remove the application or stop the service that is using ***loop*** devices.

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 deviceA 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 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 | **5096** | **19362** | **170** |
| ext2 with block size 2048 | **5008** | **19366** | **38** |
| ext2 with block size 4096 | **5024** | **19356** | **28** |
| ext2 with Journal File and block size 1024 | **5016** | **18338** | **175** |
| ext3 with block size 1024 | **5016** | **18338** | **175** |
| ext4 with block size 1024 | **5016** | **18338** | **326** |

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)?

**ext2 with block size 4096, as it has the best ratio between used blocks and total blocks. It is the best because it uses the least overhead.**

1. **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?

**to adjust various tunable filesystem parameters on ext2, ext3, or ext4 filesystems.**

1. Use **tune2fs** command with respective option to add journaling feature to ext2 file system as follows:

tune2fs -j /dev/loop0

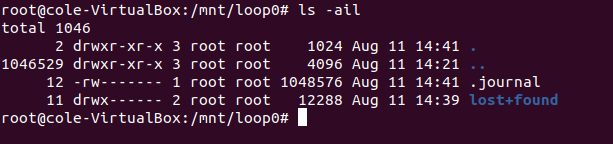
1. Use cd to change and access mounted ext2 file system on /mnt/loop0

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:

**1048576 bytes .journal**

1. Attach a screen capture that demo created journal file and respective size.



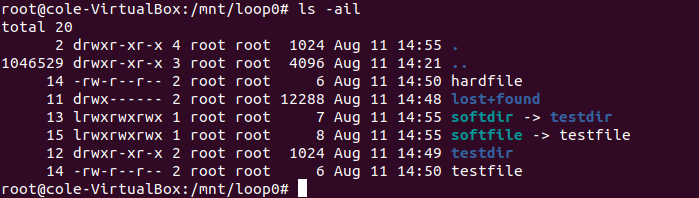
1. Use df command and analyze the file system usage. For comparison, create an ext4 file system with 1024 byte blocks. Compare **file systems usage** with file system ext2 with journaling. What is your conclusion?

**They have the same number of total blocks, but ext4 has more overhead, even with journaling on ext2**

1. **Ext Hard and Symbolic Links features \_\_\_/5**
2. Create an ext4 file system on /dev/loop0 and mount it
3. Use the command ***df*** to verify if ext4 file system was mounted
4. Access the mounted file system by using **cd /mnt/loop0**
5. Create a directory called testdir on that device (mnt/loop0)
6. Use the command touch to create an empty file called testfile in the same device (/mnt/loop0)
7. Edit testfile file and type “Hello”
8. Use the command ***info ln*** and read section 12.2 to clarify and differentiate hard links and soft links.
9. Use the command **ln –s source target** to create a soft link for both the testdir directory and the testfile file
10. 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 ?

**You cannot create a hardlink for directories because since hardlinks are viewed as the same thing in the system it could possibly create directory loops or dangling directory subtrees.**

1. Display **inodes** for the directory and file with **ls –ail**.
2. Attach screen capture that displays inodes values for the soft link and hard link and compare it with the inode of the original file (testfile). Explain results.



**The hardlink and file have the same inodes, the softlink does not.**

1. Modify the content of testfile file and compare it with the content of the hard link file.

Explain and attach respective screen captures to demo the following:

* + 1. Why is the content of the hard link file same as the original file content?

**They share the same inodes and as such are essentially the same file**

* + 1. What happens if you delete the original file while it has a hard link to it?

**You can still access and alter the hardlink**

* + 1. What happens if you delete the original file while it has a symbolic link to it?

**The softlink can no longer be accessed**

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*** command (copy files)

1. Use this program to copy the 20MB file /mydiskfile created before into a junkfile. 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**

**real 0m0.239s**

**user 0m0.004s**

**sys 0m0.015s**

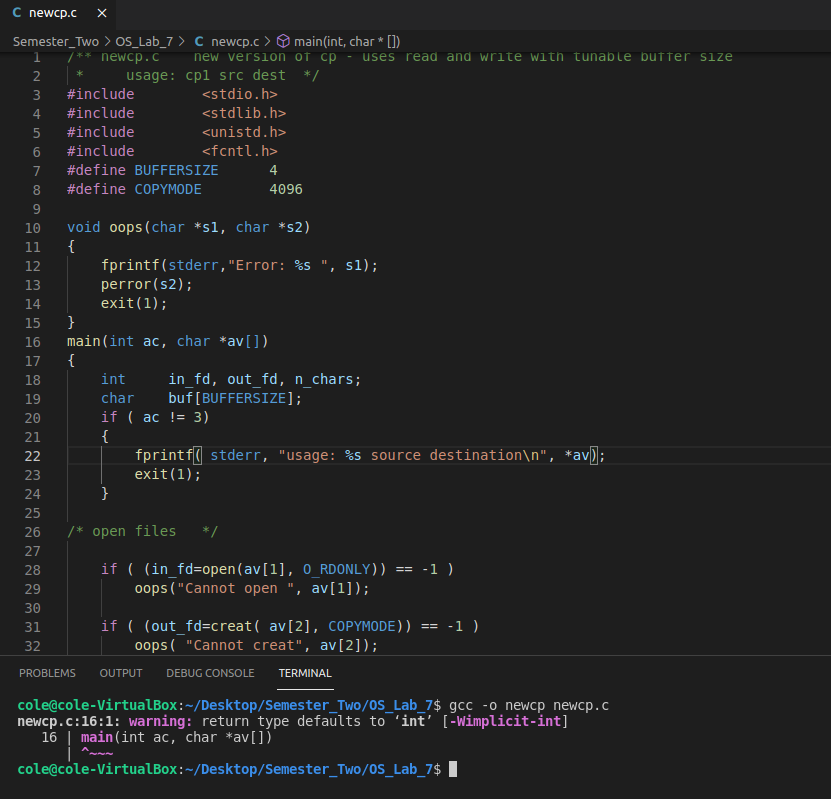
* 1. Use **strace –c ./newcp /mydiskfile** **junkfile** to identify the main system calls used in this program. Record in the following table execution time and only write() system calls required for the following buffer sizes. **Remember to compile the code after changing buffer size.**

|  |  |  |
| --- | --- | --- |
| Buffer size | Execution time (user + system) in sec. | **Only write()** system calls |
| 4096 | **0m0.019s** | **5000** |
| 2048 | **0m0.028s** | **10000** |
| 1024 | **0m0.044s** | **20000** |
| 512 | **0m0.078s** | **40000** |
| 4 | **0m8.168s** | **5120000** |

1. List the effects of decreasing the buffer size in this program.

**Decreasing the buffer size makes the program run exponentially slower**

1. Attach screen captures to demo:
   1. Code creation and compilation for 4096 buffer size



* 1. Strace for 4 buffer size

