Tutorial 3: File System Concepts (v0.11)

The purpose of this tutorial is to introduce and/or review operating systems file system concepts in the context of Linux. Complete this tutorial using your Ubuntu Virtual Machine, or another Linux system.

Tutorial Submission

This tutorial is accompanied by a Canvas quiz. While completing the tutorial below, write down your answers to each of the questions. After completing the tutorial, log into Canvas and complete the Tutorial #3 Quiz. The Canvas quiz is limited to 180 minutes. It is best to write down the answers while completing the tutorial before starting the timed quiz. Tutorial #3 provides extra credit in the "Extra Credit" category for up to (2%) extra credit for TCSS 422 in S'25.

1. Exploring the File API

Chapter 39 describes Linux C API functions to support file I/O operations. These include some that we've already discussed: open(), close(), write(), as well as additional new APIs. Review the new file APIs introduced in Chapter 39 including: lseek(), dup(), fsync(), rename(), unlink(), mkdir(), opendir(), readdir(), and closedir().

Answer the following questions related to these APIs. Create short test programs to test the functionality of the APIs as needed. Refer to the source code in chapter 39. Search the internet using a search engine such as Bing or Google to identify required C header files.

Q. What file must be included to use the readdir() API?

Create the following test program, naming the source file "wtest.c".

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
  int fd=open("foo", O_CREAT|O_WRONLY|O_TRUNC, S_IRUSR, S_IWUSR);
  for (int i=0; i<5000; i++)
  {
    write(fd, "abcde", 5);
  }
  close(fd);
}
```

Compile and run the program.

```
$gcc -o wtest wtest.c
```

Time the execution of the program using the time command.

```
$ time ./wtest
```

Q. How long does it take in milliseconds to run the original unmodified wtest.c program?

Now, immediately after the "write" command add the following line of code:

```
fsync(fd);
```

Recompile the program.
Delete the prior output file.

```
$ rm foo
```

Time the execution of the program again using the time command.

```
$ time ./wtest
```

Q. How long does it take in milliseconds to run the modified wtest.c program?

Q. If the runtime of the modified program has changed, what is likely the cause?

2. **File System Links**

A useful feature provided by Linux file systems are links. Hard links provide a direct reference to another file on the same file system.

Try creating a new file, and the creating a hard link to it.

```
echo "Hello World" > newfile

# create a hard link called 'newfile.lnk'
ln newfile newfile.lnk
```

Hard links are not allowed to link to a file or directory located on another file system. They must be on the same file system.

Symbolic links provide a mechanism to link to another file on the same file system, or on another file system. Symbolic links can also link *entire directories*.

```
# create a symbolic link for your file called 'newfile.s'
ln -s newfile newfile.s

# now compare the differences between filenames using the "stat" command
stat newfile*
```

Q. What is the difference between the inode number of the hard vs. the symbolic link?

```
# now delete the original source file (newfile)
rm newfile
```

Q. What happens to the hard link when the original linked file is deleted?

Q. What happens to the symbolic link when the original linked file is deleted?

Symbolic links are especially useful to support rapidly switching between different versions of a compiler (i.e. Java or Python) without making any system changes. This way developers can install multiple compilers and rapidly switch between different versions for different development tasks. In Ubuntu this flexibility is provided using the **update-alternatives** command. This command uses symbolic links to remap the system's active compiler.

To test this functionality, install python3.11 and python3.12 (default version for Ubuntu 24.04 LTS) on your Ubuntu VM.

```
# Install python3.11 (old version - requires external repository)
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.11

# Install the default version of python 3 from the Ubuntu package repo
sudo apt update
sudo apt install python3

# install alternatives for python
sudo update-alternatives --install /usr/local/bin/python python /usr/bin/python3.11
20
sudo update-alternatives --install /usr/local/bin/python python /usr/bin/python3 40

# show versions of python the system recognizes:
sudo update-alternatives --query python

# see which python interpreter is used by default on the command line
which python

# This should be a symbolic link, check its destination:
ls -l /etc/alternatives/python

# Test the current version of the python interpreter
python --version

# Change the current version of the python interpreter
sudo update-alternatives --config python
< select python3.11, should be option # 1 >

# test python version now:
python --version

# check symbolic link destination
ls -l /etc/alternatives/python

# Change the version of the python interpreter back to Python3
sudo update-alternatives --config python
< select python3.12, should be option # 2 and also option # 0 >

# test python version now:
python --version
```

```
# check symbolic link destination
ls -l /etc/alternatives/python
```

Q. "update-alternatives" remaps the python command to point to the selected Python interpreter. How can the selected interpreter be bypassed to run a program using python version 3.11?

Symbolic links are also very useful for extending available space in a filesystem by creating symbolic links that provide directory paths mapped to separate physical disks. A common practice is to create a symbolic link to a data directory for a relational database. This way, the root file system, which often has limited capacity, can be extended without repartitioning or reformatting the disk.

We will now demonstrate this feature by creating a virtual ramdisk and switch out the system's /tmp directory with a /tmp directory on the ramdisk.

We will install sysbench to benchmark the performance of the disk hosting the default /tmp directory.

Install sysbench as follows:

```
sudo apt install sysbench
cd /tmp
sysbench fileio prepare --file-total-size=400M
sysbench fileio --file-test-mode=rndrw --file-total-size=400M run
```

Record the Throughput results for read (MiB/s) and written (MiB/s).

Now, create a virtual ramdisk:

```
sudo mkdir /mnt/ramdisk

sudo mount -t tmpfs -o rw,size=500M tmpfs /mnt/ramdisk

Now create a tmp dir:
sudo cd /mnt/ramdisk
sudo mkdir tmp
sudo chmod a+rwx tmp
```

Backup the original /tmp directory

```
cd /
sudo mv /tmp /tmp.aside
sudo ln -s /mnt/ramdisk/tmp
sudo cd /tmp
```

Now prepare sysbench to run:

```
sysbench fileio prepare --file-total-size=400M
sysbench fileio --file-test-mode=rndrw --file-total-size=400M run
```

Record the Throughput results for read (MiB/s) and written (MiB/s).

Now inspect the /tmp symbolic link for /tmp:

```
cd /
ls -l
```

Now, restore the original /tmp

***** WARNING !! *****
THIS STEP IS EXTREMELY IMPORTANT !
FAILURE TO DO THIS STEP COULD RENDER YOUR VIRTUAL BOX VM UNBOOTABLE !!

```
sudo umount /mnt/ramdisk
sudo rm /tmp
sudo mv /tmp.aside /tmp
```

Q. Which /tmp directory had better sysbench throughput? The standard /tmp or the symbolically linked /tmp that was created on a ramdisk?

Q. Please provide /tmp throughput numbers

3. **File system journaling**

File system journaling is described in Chapter 42 as a means to recover from power loss or system crash. Journaling is an improvement over the original fsck (file system checker) method. Sections 42.2 (fsck) and 42.3 (journaling) in the textbook describe these two mechanisms. In general, a journaling file system tracks changes not yet physically written to the disk (media) by recording the operations to be performed in a data structure called a "journal". Incomplete file writes can be stored in the log and completed or aborted when power is restored to the computer. The default filesystem in Ubuntu 24.04 is the ext4 journaling file system. By default, journaling is enabled on any ext4 filesystem created under Ubuntu 24.04.

Next, we will see how Linux block devices can allow us to experiment with file systems to run performance experiments.

We can create a virtual disk by mounting a formatted binary data file in Linux.

First, let's create two binary data files using the dd (data duplicator) command. We user the /dev/zero device to copy a stream of zeros to the file.

```
# return to the user's home directory
cd

# create a directory for disk images
mkdir disks
cd disks
sudo dd if=/dev/zero of=disk1.img bs=100M count=1
sudo dd if=/dev/zero of=disk2.img bs=100M count=1

# associate a virtual loop back device with the new binary data files
sudo losetup -fP disk1.img
sudo losetup -fP disk2.img

# find the identities of the new loop back devices
# the names will be "/dev/loop" followed by a number.
# look for the entries for disk1.img and disk2.img
sudo losetup -a
```

```
# using the loop back device identities, format the new virtual disks,
# here we will assume the loop back identities are /dev/loop7 and /dev/loop8
# we will format one image with the ext2 file system, and one with the ext4 file system
sudo mkfs.ext2 /dev/loop7   # for disk1.img
sudo mkfs.ext4 /dev/loop8   # for disk2.img
# next, create directories to mount these file systems
sudo mkdir /mnt1
sudo mkdir /mnt2

# mount the new file systems
sudo mount /dev/loop7 /mnt1
sudo mount /dev/loop8 /mnt2

# now prepare to run sysbench on the ext2 filesystem
sudo bash
cd /mnt1
# benchmark and write down the throughput results for the ext2 file system:
sysbench fileio prepare --file-total-size=80M
sysbench fileio --file-test-mode=rndrw --file-total-size=80M run

# now prepare to run sysbench on the ext4 filesystem
cd /mnt2

# benchmark and write down the throughput results for the ext4 file system:
sysbench fileio prepare --file-total-size=80M
sysbench fileio --file-test-mode=rndrw --file-total-size=80M run
```

Now disable journaling on the ext4 filesystem.
This requires unmounting the filesystem:

```
# replace the loop device name as needed to match your ext4 loop device name
sudo umount /mnt2
tune2fs -O ^has_journal /dev/loop8

# remount the ext4 filesystem now without journaling
mount /dev/loop8 /mnt2

# repeat the sysbench test and write down the throughput results for ext4:
cd /mnt2
sysbench fileio prepare --file-total-size=80M
sysbench fileio --file-test-mode=rndrw --file-total-size=80M run
```

Q. Which filesystem had the highest read/write throughput results measured in MiB/s?

Q. Please report your read/write throughput values for all three tests:
ext2:
ext4 with journaling:
ext4 without journaling:

4. **Inodes**

Filesystems consist of individual inodes which are records used to track individual files that are stored on the disk. When formatting a disk with ext2 or ext4, a default inode to disk space ratio is used to automatically

create a specific number of inodes. Let's now create a new binary disk image and pay careful attention to the inode ratio when formatting the disk:

```
# return to the user's home directory
cd

# return to the disk images directory
cd disks

# create a new image
sudo dd if=/dev/zero of=disk3.img bs=100M count=1


# associate a virtual loop back device with the new binary data file
sudo losetup -fP disk3.img

# find the identity of the new loop back device
# the names will be "/dev/loop" followed by a number.
# look for the disk3.img entry
sudo losetup -a

# using the loop back device identities, format the new virtual disks,
# here we will assume the new loop back identity is /dev/loop9
# we will format using the ext4 file system
sudo mkfs.ext4 /dev/loop9   # for disk3.img
sudo mkdir /mnt3
sudo mount /dev/loop9 /mnt3
```

The output from mkfs.ext4 should report the number of inodes created.
This disk image should be 100 MB.

Q. One inode is created for every storage block on the disk. How large is this storage block size in KB for which an inode is created?

Q. How many inodes are created?

Using the "sysbench fileio" benchmark, prepare, but do not run the benchmark:

```
cd /mnt3
sudo sysbench fileio prepare --file-total-size=80M
```

Now, write down the % of available inodes on the /mnt3 file mount:

```
df -i
```

Q. What % of inodes are in use initially on the /mnt3 file system mount?

Now, let's change the default inode ratio.
Reformat disk3.img:

```
cd /
sudo umount /dev/loop9
sudo mkfs.ext4 -i 102400 /dev/loop9
sudo mount /dev/loop9 /mnt3
cd /mnt3
sudo sysbench fileio prepare --file-total-size=80M
```

Now, write down the % of available inodes on the /mnt3 file mount:

```
df -i
```

Q. What % of inodes are in use with the revised inode ratio on the /mnt3 file system mount?

To complete this assignment, to obtain credit, log in to Canvas, and complete the **Tutorial 3 Quiz**.

Version	Date	Change
0.1	6/03/2025	Original Version
0.11	6/09/2025	Improved fsync test procedure. Updated python versions for Ubuntu 24.04

Please help improve this tutorial by reporting any errors or discrepancies found.