Results

Sahil Kasturi

This assessment has been regraded. Questions 8, 19 have been affected.

89.66%

26
Out of 29 points

01:15:02
Time for this attempt

Your Answers:

1 0 / 0 points

Read the section titled Pseudo Filesystems in Learning Modern Linux and answer the following questions.

- What is the PID of the shell in your GCP VM. Use the **ps** command to find the PID of the shell in which you ran the command.
- How many FD's are in the shell's file descriptor table? Use the **/proc** filesystem.

To find the PID of my shell I ran:
echo $$

This showed the PID of the shell process I was using. Then, to check how many file descriptors (FDs) were open for that shell, I used:
ls /proc/$$/fd | wc -l

This showed there were 3 file descriptors, which are the standard ones — stdin (0), stdout (1), and stderr (2).

The /proc directory is part of Linux's pseudo-filesystem, which exposes kernel and process information as files. So /proc/<PID>/fd lets you see all the file descriptors currently open by that process.

Correct

2 0 / 0 points

Use **mount** to find the device on which the root *ext4* filesystem is laid out, and then use **fdisk -l** to determine the block size of the file system.

To find the device where the root (/) filesystem is located, I used the command "mount | grep ' / '". The output showed that the root ext4 filesystem is on /dev/sda1. After identifying the device, I used "sudo fdisk -l /dev/sda" to display detailed information about the disk, including its block size. The output showed a sector size of 512 bytes (logical) and 4096 bytes (physical), meaning the filesystem's block size is 4096 bytes, or 4 KB.

Correct

3 0 / 0 points

Find how long the Ubuntu OS on your GCP VM has been up since its last reboot by querying the **/proc** filesystem.

By checking the `/proc/uptime` file on my Ubuntu VM, I found the first value to be 96.80 seconds. This indicates that my system has been running for approximately 1 minute and 37 seconds since its last reboot.

Correct

4 0 / 0 points

Print "Hello World" on your terminal by writing "Hello World" to the *file* that the terminal device represents.

I first ran the tty command to determine my terminal device, which returned /dev/pts/0. Then I used the command echo "Hello World" > /dev/pts/0 to write the message directly to that terminal device file. This successfully printed "Hello World" on my terminal.

Correct

5 2 / 2 points

What is the significance of the **x** bit (execute) on a directory in Linux? Create a directory, say **f**, and create three files inside **f**: say these files are *a*, *b*, *c*. Change the permission of **f** to *400* using **chmod 400 f**. Can you still do **ls f**? Can you do **ls -l f**? Explain what you see.

The execute (x) bit on a directory controls whether users can enter or traverse that directory and access files within it. To test this, I created a directory named f and three files inside it (a, b, and c). Then, I changed the directory's permissions using chmod 400 f, which removed the execute permission. After this, running ls f still worked because the read (r) permission allows listing the names of files inside the directory. However, when I ran ls -l f, it failed to show details for each file and displayed "Permission denied," because listing file details requires execute permission to access metadata about the files inside the directory. Without the execute bit, I also could not cd into f or open any files within it.

Correct

6 2 / 3 points

Interpret the following mode bits in octal on a filesystem object. What type of object is it?

- 0100664.
- 010664.
- 0101664.

0100664 : regular file (0100000 type), permissions = rw-rw-r-- (0664)

0100664 : regular file (0100000 type), permissions = rw-rw-r-- (0664)

0101664 : regular file (0100000 type) with sticky bit set (01000), permissions = rw-rw-r-T (0664 + sticky bit)

Graded

7 2 / 2 points

Do problem 1 under In-class → Filesystems on the class website.

When the program runs, it first writes the characters "pq" to foo, then writes "jklmn" plus the null terminator \0, so the final contents of foo are pqjklmn\0 (8 bytes). Viewing it with od -c -A d foo shows:

```
0000000 p q j k l m n \0
```

```
0000008
```

If foo did not exist, it is created with permissions 0700 (rwx------), determined by S_IRWXU and the process's umask. If foo already existed, its previous permissions remain unchanged, but the contents are truncated and replaced with the new data.

Correct

This question has been regraded.

8 Previous score 0 / 2 points Regrade score 0 / 2 points

Do Problem 4 under In class -> Filesystems on the class website

Answer: ✗ ---r-xr-x (octal 055)

Correct Answer: **055, 0100555, 0055**

9 2 / 2 points

The setuid and setgid bits. The **setuid** (set user ID) and **setgid** (set group ID) bits are special permission bits in Linux that affect how processes execute:

- **Setuid bit (octal 4000)**: When set on an executable file, the process runs with the privileges of the file's owner rather than the user who executed it. For example, `/usr/bin/passwd` has the setuid bit set, allowing regular users to change their passwords (which requires modifying `/etc/shadow`, normally restricted to root).
- **Setgid bit (octal 2000)**: When set on an executable, the process runs with the privileges of the file's group. When set on a directory, new files created within inherit the directory's group ownership rather than the creator's primary group.

These bits appear in the execute position: `s` for setuid/setgid. The uppercase letter S indicates the corresponding execute bit is not set. In other words,

$$s = x + setuid$$

while

$$S = \bar{x} + setuid$$

What is the octal representation of the file permissions `-rwsr-xr-x`?

- ◯ 0755
- ✓ ● 4755
- ◯ 2755
- ◯ 6755

10 2 / 2 points

Consider the following program.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

int main(void) {
  if(open("/foo/bar", O_RDONLY) < 0) exit(1);
}
```

The program exits with a value of 1 if it cannot open the file **/foo/bar**. How can we change this program to print a more specific message that indicates the reason for failure to open the file? Hint: use **perror** (man 3 perror).

You can use perror() right after the failed open() call to print the system-generated error message.

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

```
#include <fcntl.h>

#include <stdlib.h>



int main(void) {

if (open("/foo/bar", O_RDONLY) < 0) {

perror("open"); // prints "open: <reason>" (e.g., "open: No such file or directory")

exit(1);

}

}
```

Correct

11 2 / 2 points

Which bit pattern represents a directory with permissions `drwxr–xr–x`? Note that the leading 0 in the choices indicates that the # is specified in octal. You can use an online octal to binary converter such as the one at www.rapidtables.com to simplify your life.

- [✓ 0040755]
- 0100755
- 0040644
- 0120755

12 2 / 2 points

Consider this code that creates a file:

```
int fd = open("newfile.txt", O_CREAT | O_WRONLY, 0644);
```

If the process's umask is set to *0022*, what will be the actual permissions of the created file? Explain your reasoning. See APUE Section 4.8 for an explanation of umask. Specifically, for the **open** system call, the actual permission bits used to create the new file are

$$mode \wedge \overline{umask}$$

open(..., 0644) creates the file with mode & ~umask. With umask 0022, ~0022 = 0755, so:

0644 & 0755 = 0644 → final perms 0644 (-rw-r--r--).

Correct

13 2 / 2 points

A file has permissions `-rwsr-sr-x` with octal 6755. What happens when a regular user executes this file? You can read more about setuid and setguid in APUE Section 4.4.

- ○ The process runs with the user's normal privileges
- [✓ The process runs with the owner's user ID and the file's group ID]
- ○ The process runs with root privileges
- ○ The execution is denied due to security restrictions

14 0 / 0 points

What are all the *groups* that you belong to on your GCP VM? Run the **id** command. How many groups are present on your VM? All the groups are defined in the **/etc/group** file. Read more about group id's in APUE Section 6.5.

I belong to the following groups on my GCP VM: sahilkasturi2004, adm, dialout, cdrom, floppy, audio, dip, video, plugdev, netdev, lxd, ubuntu, and google-sudoers.

There are 13 groups in total.

All of these groups are defined in the /etc/group file, which lists group names and their corresponding group IDs used by the system.

Correct

15 0 / 0 points

What does the library function **makecontext** (man 3 makecontext) do? Summarize in a couple of sentences. Don't copy the description from the manual page.

makecontext() sets up a user-level context so that when it's later activated with setcontext() or swapcontext(), it begins executing a specific function with given arguments. In simpler terms, it lets you define what function a thread or coroutine should start running next, allowing you to manually control execution flow between multiple contexts.

Correct

16 2 / 2 points

Given the following code snippet, what will be printed? Read the man page for the system call **stat** in section 2 of the man pages. You can find more about **S_ISREG** in APUE Section 4.3.

```
struct stat sb;
stat("/usr/bin/sudo", &sb);

if (S_ISREG(sb.st_mode)) {
    printf("Regular file\n");
}
if (sb.st_mode & S_ISUID) {
    printf("Setuid bit is set\n");
}
```

Regular file

Setuid bit is set

Correct

17 2 / 2 points

Analyze this code that modifies file permissions:

```
struct stat sb;
stat("myfile", &sb);
chmod("myfile", sb.st_mode | S_IXUSR | S_IXGRP);
```

What will happen to the file's permissions? Read the manual page for the system call **chmod** in section 2.

This code first retrieves the current permissions of myfile using stat(), then calls chmod() to add execute (x) permission for the file's owner and group.

In effect, whatever permissions myfile already had remain the same, but now both the user (owner) and group will have execute permission. For example, if it was originally rw-r--r-- (0644), it becomes rwxr-xr-- (0754).

Correct

18 2 / 2 points

A directory has permissions `drwxrwsr-x` (octal 2775). When a user creates a file in this directory, what group will own the new file? See Resources → Filesystems → u+/g+/t for directories on the class website.

- The user's primary group
- [✓ The directory's group]
- The root group
- A new group is created automatically

This question has been regraded.

19 Previous score 0 / 2 points Regrade score 2 / 2 points

Which of the following file mode bits has the value **0001** in octal?

○ Owner execute permission

○ Group execute permission

✓ ● Other execute permission

○ Sticky bit

20 0 / 0 points

Consider a file with permissions `-rwsr-Sr-x` (note the capital 'S'). What does the capital 'S' indicate, and what security implications might this have?

> The capital 'S' means the setgid bit is set but the group execute bit is not set. In -rwsr-Sr-x, the file is setuid (user s), setgid (S), but group lacks execute. Security-wise, if the file is executable (here via owner or other x), running it will still get the file's effective group ID (setgid). That can grant unintended group privileges—especially since it's world-executable—so this combo often signals a misconfiguration and should be reviewed.

Correct

21 2 / 2 points

Examine the following code snippet, and read the manual page for the **stat** system call.

```
struct stat sb;

stat("/home/user/myfile", &sb);
if (sb.st_mode & S_IRUSR) {
    printf("Owner has read permission\n");
}
```

What does the S_IRUSR constant represent, and what operation is being performed with the & operator?

> S_IRUSR is a constant that represents the owner's read permission bit (0400 in octal). In the code, the & operator is used to test whether that bit is set in sb.st_mode. If the result is nonzero, it means the file's owner has read permission, so the program prints "Owner has read permission."

Correct