CMP325
Operating Systems
Lecture 29

File Permissions

Fall 2021

Arif Butt (PUCIT)

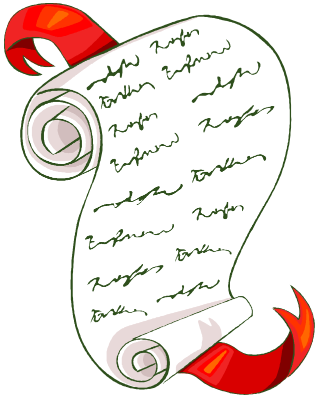
Note:

Some slides and/or pictures are adapted from course text book and Lecture slides of

- Dr Syed Mansoor Sarwar
- Dr Kubiatowicz
- Dr P. Bhat
- Dr Hank Levy
- Dr Indranil Gupta

For practical implementation of operating system concepts discussed in these slides, students are advised to watch and practice video lectures on the subject of **OS with Linux** by Arif Butt available on the following link:
http://www.arifbutt.me/category/os-with-linux/

Today's Agenda



- Overview of Protection & Security
- Protection in Linux
- How Permissions are managed
- Default Permissions
- Changing Permissions
  - Symbolic Way
  - Octal Way
- Special Permissions
  - SUID bit
  - SGID bit
  - Sticky bit

Overview of File Protection

PROTECTION AND SECURITY

- When information is kept in a computer system, we want to keep it safe from physical damage (reliability) and improper access (protection).
- File owner/creator should be able to control:
  - What can be done to the file / directory
  - By whom it can be done
- Different OS support different types of access to files and directories:
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List
- The purpose of protection system is to prevent accidental or intentional misuse of a system

PROTECTION AND SECURITY

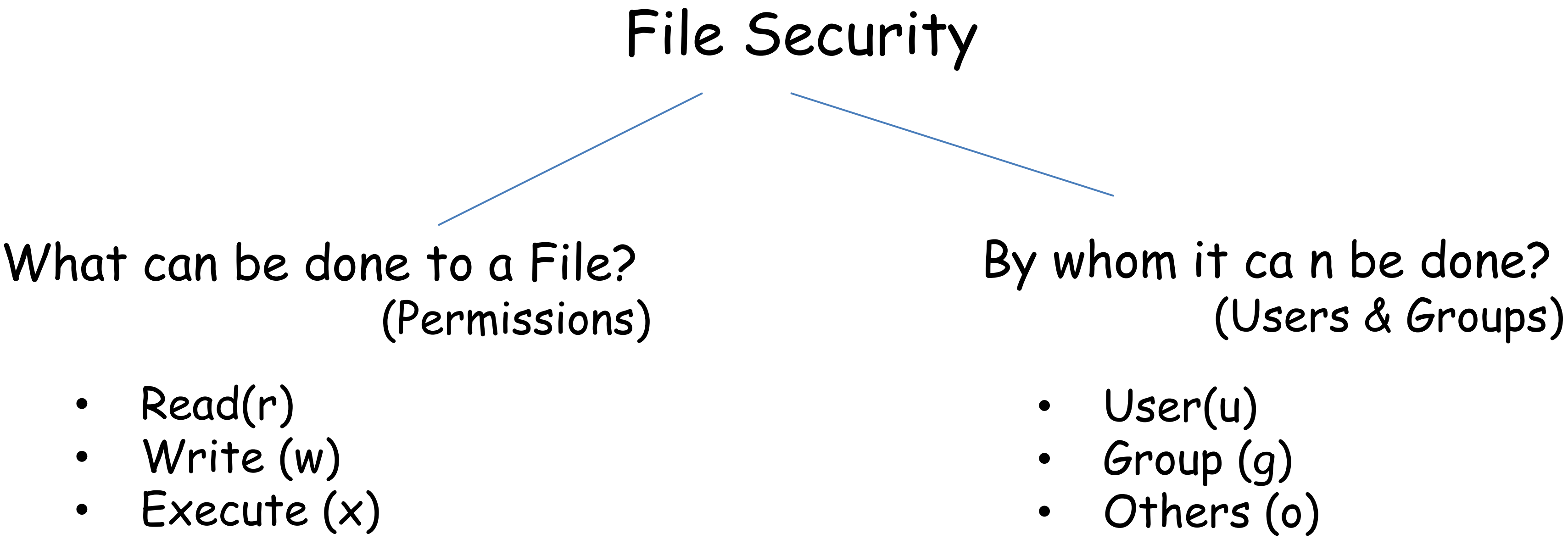
Three aspects to a protection mechanism:

- **Authentication** (Who goes there?): Authentication means who is allowed to access the system. Any one (even a program) who wants to access a system needs to login and for that he/she must have a proper user account on it. For example in a Linux based machine every user must have an entry in the local user data base file /etc/passwd along with /etc/shadow, /etc/gourp and /etc/gshadow
- **Authorization** (Are you allowed to do that?): means the authority of a user to perform various operations
- **Accountability**: means after a user has been successfully authenticated and is authorized as well to perform a specific task, even after that the foot print of the user are recorded for later forensic usage

PROTECTION IN LINUX

- **Users** - Every user of a system is assigned a unique UID. User's names and UIDs are stored in /etc/passwd file. Users cannot read, write or execute each others files without permissions
- **Groups** - Users are assigned to groups with unique GID. GIDs are stored in /etc/group. Each user is given his own private group by default. He / she can belong to other groups to gain additional access. All users in a group can share files that belong to that group.
- Three Classes of Users
- **User / owner** - The owner is the user who created the file. Any file you create, you own
- **Group** - A user / owner of a file can grant access of a file to the members of a designated group
- **Others** - A user / owner of a file can also open up access of a file to all other users on the system

PROTECTION IN Linux



File Security

What can be done to a File?
(Permissions)

- Read(r)
- Write (w)
- Execute (x)

By whom it ca n be done?
(Users & Groups)

- User(u)
- Group (g)
- Others (o)

PROTECTION IN LINUX

Read Write and Execute permissions have different meanings for files and directories:

For files, the permissions have following meanings:

READ: Enables users to open files and read its contents using; less, more, head, tail, cat, grep, sort, view

WRITE: Enables users to open a file and change its contents using vi, vim, peco, nano

EXECUTE: Enables users to execute files as commands

For directories the permissions have following meanings:

READ: Users can list directory contents using ls command

WRITE: Users can create, delete files (that he owns) in the directory using mkdir, touch, cp commands

EXECUTE: Users can search in the directory and change to it using the cd command. Without execute permissions on a directory, read/write permissions are meaningless

PROTECTION IN UNIX

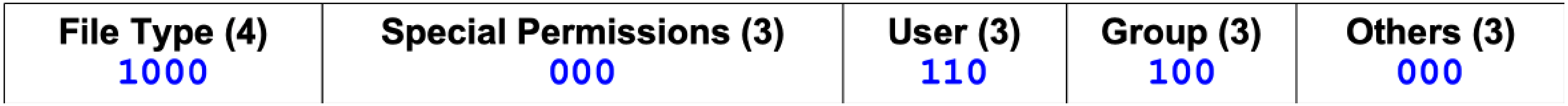
- Every file / directory contains a set of permissions that determine who can access them and how. Lets view the permissions associated with the /etc/passwd file

```
#ls -l /etc/passwd
-rw-r--r--   1     root  root 695   Dec 7 12:48  passwd
```

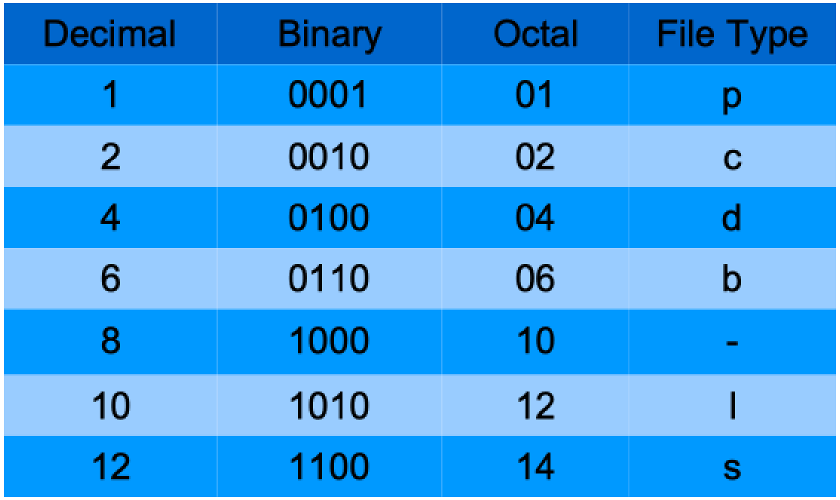
- Type of file (-, d, l, p, c, b, s). Once created cannot be changed
- Permissions (rwx, rwx, rwx). Can be changed using umask(1)
- Link Count (Number of hard links to this file). Can be changed using ln(1)
- Owner. Can be changed using chown(1)
- Group. Can bbe changed using chgrp(1)
- Size. Can be changed by changing contents of file
- Date/Time (modification time, access time, status change time)
- Name

PROTECTION IN UNIX

- Whenever a user access a file / directory, the permissions are applied in following fashion:
  - If you are the user/owner, the user/owner permissions apply
  - If you are in the group, the group permissions apply
  - If you are neither the owner nor the group member, then others permissions apply
- To maintain the file type and permissions, all UNIX based systems use a 16 bit architecture as shown:



File Type (4)	Special Permissions (3)	User (3)	Group (3)	Others (3)
1000	000	110	100	000



Decimal	Binary	Octal	File Type
1	0001	01	p
2	0010	02	c
4	0100	04	d
6	0110	06	b
8	1000	10	-
10	1010	12	l
12	1100	14	s

DEFAULT PERMISSIONS

- The new permissions on a file are set by the creator program like vim(1), mkdir(1)

```
open(“f1.txt”, O_CREAT| O_RDWR, 0666);
mkdir(“d1”, 0777);
```

- The final permissions on files are

```
mode & ~umask
```

- The final permissions on directories are

```
mode & ~umask & 0777
```

- To display the current umask value use the umask(1) command, which can also be used to change the umask value

```
$ umask
$ umask 077
```

CHANGING PERMISSIONS ON FILES

- The access rights for any given file can be modified by using the change mode (chmod) command. chmod takes two lists as its arguments: permission changes and filenames.

```
chmod mode filename/dirname
```

- We can use two different modes
  - Symbolic
  - Octal

SYMBOLIC METHOD OF CHANGING PERMISSIONS

- Symbols for Level

  u - Owner of a file

  g - Group to which the user belongs

  o - All other users

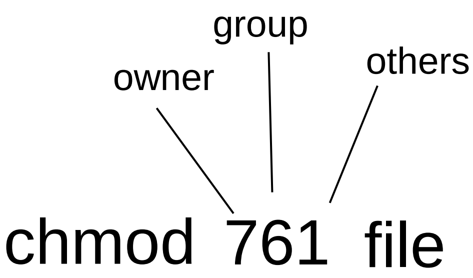
  a - All Can replace u, g, or o
- Symbols for Permissions

  + Add the following permissions (does'nt affect other pmns).

  - Remove the following permissions (does'nt affect other pmns).

  = Assigns entire set of permissions.
- Examples
  - chmod u=rwx filename
  - chmod g=rx filename
  - chmod g+x filename
  - chmod o-w filename
  - chmod a+r filename
  - chmod a+r-x filename
  - chmod o+r-wx filename
  - chmod g=rw,o-w filename

OCTAL METHOD OF CHANGING PERMISSIONS



group

owner

others

chmod 761 file

- With the chmod command, we can use a three digit octal number as mode. Using octal numbers all permissions are completely reset. You can't add/remove individual settings, as we can do in symbolic method of changing permissions
- The three categories, each with three permissions, conform to an octal binary format. Octal numbers have a base 8 structure. When translated into a binary number, each octal digit becomes three binary digits. Three octal digits in a number translate into three sets of three binary digits, which is nine altogether— and the exact number of permissions for a file.
- The first octal digit applies to the owner category, the second to the group, and the third to the others category. The following table explains it.

r w x	Level	Permissions
0 0 0	0	No permissions.
0 0 1	1	Execute only
0 1 0	2	Write only
0 1 1	3	Write & Execute
1 0 0	4	Read only
1 0 1	5	Read & Execute
1 1 0	6	Read & Write
1 1 1	7	Read, Write & Execute

SPECIAL PERMISSIONS

SUID bit: (Set-User-ID bit)

- SUID bit is normally set for executable programs
- If a program has this bit set, it executes with the power of the owner of the program
- It is represented by an **s** in the execute portion of owner permissions, or a capital **S** in case if the owner execute permission is off
- The passwd program has its SUID bit set and that is how using passwd program one can write the /etc/shadow file owned by root
- To check the executable files in our system having their SUID bit set, run the following command

```
# find  /  -perm   /4000
```

- To set the SUID bit of a program

```
# chmod  u+s myexe
```

- SUID bit has no meanings on a directory

SPECIAL PERMISSIONS

SGID bit: (Set-Group-ID bit)

- SGID bit is set for executable programs and directories
- If a program has this bit set, it executes with the power of the group of the program
- It is represented by an **s** in the execute portion of group permissions, or a capital **S** in case if the group execute permission is off
- The chage program has its SGID bit set and that is how using chage program one can write the /etc/shadow file
- To check the executable files in our system having their SGID bit set, run the following command

```
# find  /  -perm   /2000
```

- To set the SGID bit of a program

```
# chmod  g+s myexe
```

- SGID bit on directories are used in a shared group environment. Ay file created in a directory with SGID bit set will automatically inherit the group membership of that directory

SPECIAL PERMISSIONS

Sticky bit: (On Directories)

- In a shared group environment, all the members should have read as well as write permissions on directories to create files in that shared directory
- But setting these permissions on a directory, let all the group members delete each other files as well, which is not required. Solution is sticky bit
- If a directory having rwx permissions to all has this bit set, no one can delete each others file
- It is represented by an **t** in the execute portion of others permissions, or a capital **T** in case if the others execute permission is off
- The /tmp directory has its sticky bit set and that is how although being shared among all users of the system, no one can delete each others files
- To check the all the directories in our system having their sticky bit set, run the following command

```
# find   /  -perm    /1000
```

- To set the SGID bit of a program

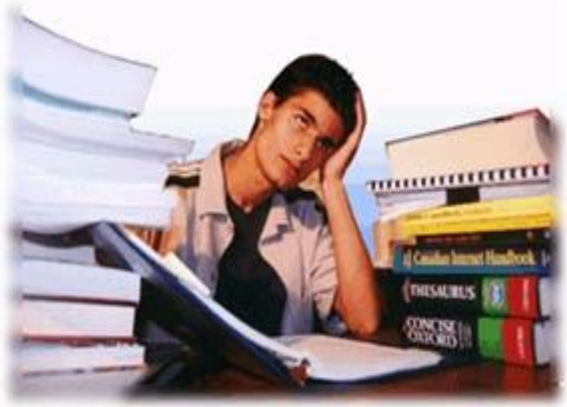
```
# chmod  o+t mydir
```

SPECIAL PERMISSIONS

Sticky bit: (On Files)

- On older UNIX implementations, sticky bit was provided as a way of making commonly used programs run faster
- The underlying concept was " Loading a program from disk (where it may be fragmented) is slow as compared to loading a program from swap space on disk (where it is not fragmented)
- If the sticky bit of an executable is set then the first time it is executed, a copy of the program text is saved inn the swap area, thus it sticks on the swap space and loads faster on subsequent execution
- But today, the current sophisticated memory management schemes have rendered this use of stick bit obsolete

We're done for now, but Todo's for you after this lecture…



- Go through the related video lectures # 22 and 23.
- Practice, practice and practice…