Lab 3: Computer Architecture: Users, files, processes

Walkthrough video:

Computer Architecture: Users, files, processes 3-1 https://www.youtube.com/watch?v=qdEr98Lqak0

Learning Objectives

1. Explore various aspects of computer architecture such as processes, file systems, users and access control

Technologies Covered

- Windows, MacOS, Linux
- Bash, PowerShell, Command Prompt

1. Files, processes and others

1.1. Command Line

Operating systems offer a number of different ways of interacting with files, processes and other aspects of the system through the use of commands typed in a terminal window. These commands can even be added to files and run as scripts to perform more complicated actions.

On Windows, there is a program called the Command Prompt as well as a program called PowerShell, whilst on the Mac and Linux there is the Terminal program that runs various types of "shell" programs. Many of the commands in these systems are similar and so ***pwd*** (print working directory) will print the current directory

We are going to stick with Bash which runs on Windows (WSL), Mac OS and Linux. To do this, run the command:

```
sudo docker run -it --rm uwacyber/cits1003-labs:bash
```

```
root@2577070501db:/# pwd
/
```

The pwd command shows that we are in the root directory and when we list the contents of that directory using the `ls -al` command, we will get:

```
root@2577070501db:/# ls -al
total 60
drwxr-xr-x   1 root root 4096 Jul 13 05:58 .
drwxr-xr-x   1 root root 4096 Jul 13 05:58 ..
-rwxr-xr-x   1 root root    0 Jul 13 05:58 .dockerenv
lrwxrwxrwx   1 root root    7 Jun  9 07:27 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 15  2020 boot
drwxr-xr-x   5 root root  360 Jul 13 05:58 dev
drwxr-xr-x   1 root root 4096 Jul 13 05:58 etc
drwxr-xr-x   2 root root 4096 Apr 15  2020 home
lrwxrwxrwx   1 root root    7 Jun  9 07:27 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Jun  9 07:27 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Jun  9 07:27 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Jun  9 07:27 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Jun  9 07:27 media
drwxr-xr-x   2 root root 4096 Jun  9 07:27 mnt
drwxr-xr-x   2 root root 4096 Jun  9 07:27 opt
dr-xr-xr-x 222 root root    0 Jul 13 05:58 proc
drwx------   2 root root 4096 Jun  9 07:31 root
drwxr-xr-x   5 root root 4096 Jun  9 07:31 run
lrwxrwxrwx   1 root root    8 Jun  9 07:27 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Jun  9 07:27 srv
dr-xr-xr-x  13 root root    0 Jul 13 05:58 sys
drwxrwxrwt   1 root root 4096 Jul 13 05:54 tmp
drwxr-xr-x  13 root root 4096 Jun  9 07:27 usr
drwxr-xr-x   1 root root 4096 Jun  9 07:31 var
```

All operating systems have file systems that operate on the basis of a hierarchy of directories or folders. We used the ls command to list the contents of the root directory `/`. The flags (arguments) `-al` passed to the ls command means to list all the contents and in a long version. The layout here is a typical format of a Linux file system. We will go through what some of these directories are for but in summary:

/ is the root directory and only the user root has access to write in this directory. The user root's home directory is `/root`.

to **/usr/bin**

/sbin contains system binaries like `iptables`, `reboot`, `fdisk`, `ifconfig`, etc.

/etc contains configuration files and scripts for services running on the system. Also contains the `passwd` and shadow files that contain user and password information.

/dev contains device files that are the interface with physical devices on, or attached to, the system such as tty devices `/dev/tty1`.

/proc contains files that store information about system processes like uptime for example.

/var contains files like logs (`/var/logs`), backups (`/var/backups`), mail (`/var/mail`) and spool (`printing; /var/spool`). There is also a `/var/tmp` directory that can be used to run programs out of. This directory does survive reboots however. The directory `/var/www/html` is often used as the root directory of the web server.

/tmp contains temporary files as mentioned previously. Files get deleted on reboot.

/usr contains user binaries, libraries, documentation and source code

/usr/local contains users programs that you install from source.

/home contains user home directories

/boot contains boot loader files

/lib contains system libraries

/opt contains optional add-on applications

/mnt is a location for mounting temporary filesystems

/media is a location for mounting removable media devices like CDs

/srv contains specific service related data

Users have a ***home directory*** which in Windows is usually located in c:\Users, on the Mac it is `/Users` and on Linux it is in `/home`. On this container, it will be empty because we only have one user, root, whose home directory is /root

Navigating around can be done using the `cd` (change directory) command with an argument that tells cd which directory you want to move to. Two special shortcuts are the "." (single dot) and ".." (double dot) that specify the current directory and the parent directory respectively.

```
root@2577070501db:/# cd /home
root@2577070501db:/home# cd ..
root@2577070501db:/#
```

Creating, deleting, copying and moving a file

```
root@2577070501db:/# cd /root
root@2577070501db:~# touch file.txt
root@2577070501db:~# ls
file.txt
root@2577070501db:~# cp file.txt file2.txt
root@2577070501db:~# ls
file.txt  file2.txt
root@2577070501db:~# rm file2.txt
root@2577070501db:~# mv file.txt file2.txt
root@2577070501db:~# ls
file2.txt
root@2577070501db:~# rm file2.txt
```

> If you ever want help with a command, you can type ***help <command>*** on Windows or ***man <command>*** on Linux or Mac.

Finding files

Linux has a find command that can be used for finding files. The basic syntax is:

```
find <starting directory> -name <file to find>
```

find will look in all sub-directories and report on any files that match the name provided

```
root@2577070501db:~# find / -name ls
/usr/bin/ls
```

We can use wildcard characters as follows:

```
root@2577070501db:/# find / -name host*
/etc/hosts
/etc/host.conf
/etc/hostname
/usr/share/doc/hostname
/usr/share/vim/vim81/syntax/hostsaccess.vim
/usr/share/vim/vim81/syntax/hostconf.vim
/usr/share/vim/vim81/ftplugin/hostsaccess.vim
```

```
/usr/share/nmap/scripts/hostmap-robtex.nse
/usr/share/nmap/scripts/hostmap-bfk.nse
/usr/bin/hostname
/usr/bin/hostid
/sys/class/scsi_host/host0
/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0004:00/VMBUS:00/fd1d2cbd-ce7c-535c-966
/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0004:00/VMBUS:00/fd1d2cbd-ce7c-535c-966
/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0004:00/VMBUS:00/fd1d2cbd-ce7c-535c-966
/sys/devices/LNXSYSTM:00/LNXSYBUS:00/ACPI0004:00/VMBUS:00/fd1d2cbd-ce7c-535c-966
/sys/bus/scsi/devices/host0
/proc/sys/kernel/hostname
/var/lib/dpkg/info/hostname.md5sums
/var/lib/dpkg/info/hostname.list
```

1.2. Hidden Files and other Attributes

In Linux, files that start with a period (.) are hidden from the directory listing command ls. To see them, you need to use the `-a` flag:

```
root@2577070501db:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  roo
root@2577070501db:/# ls -al
total 60
drwxr-xr-x   1 root root 4096 Jan 25 06:24 .
drwxr-xr-x   1 root root 4096 Jan 25 06:24 ..
-rwxr-xr-x   1 root root    0 Jan 25 06:24 .dockerenv
lrwxrwxrwx   1 root root    7 Jan  5 16:47 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 15  2020 boot
drwxr-xr-x   5 root root  360 Jan 25 06:24 dev
drwxr-xr-x   1 root root 4096 Jan 25 06:24 etc
drwxr-xr-x   2 root root 4096 Apr 15  2020 home
lrwxrwxrwx   1 root root    7 Jan  5 16:47 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Jan  5 16:47 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Jan  5 16:47 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Jan  5 16:47 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Jan  5 16:47 media
drwxr-xr-x   2 root root 4096 Jan  5 16:47 mnt
drwxr-xr-x   2 root root 4096 Jan  5 16:47 opt
dr-xr-xr-x 217 root root    0 Jan 25 06:24 proc
drwx------   1 root root 4096 Jan 25 04:28 root
drwxr-xr-x   5 root root 4096 Jan  5 16:50 run
lrwxrwxrwx   1 root root    8 Jan  5 16:47 sbin -> usr/sbin
-rwxr-xr-x   1 root root   84 Jan 25 04:27 script.sh
drwxr-xr-x   2 root root 4096 Jan  5 16:47 srv
```

```
                                                              .
drwxr-xr-x  13 root root 4096 Jan  5 16:47 usr
drwxr-xr-x  11 root root 4096 Jan  5 16:50 var
root@2577070501db:/#
```

For the above example, the `.dockerenv` is the hidden file.

Linux has a limited set of specific attributes on a file that controls how the file is accessed. One attribute for example is the *Append Only* attribute that only allows the write operations on the file to append to it and not overwrite any existing content. Another attribute is Immutable which does not allow the file contents or metadata to change at all. You can list and change attributes on Linux using *`lsattr`* ``` and _ chattr _ ``` programs

1.3. Downloading Files

Question 1. Find the file

In the `/root` directory, create a directory called `cits1003`. In that directory, create a subdirectory called `lab3`. Then, use the `cd` command to get into this directory and then use the command `wget` to download the file located at "https://github.com/opentrace-community/opentrace-ios/archive/refs/heads/master.zip"

```
wget https://github.com/opentrace-community/opentrace-ios/archive/refs/heads/mas
```

```
--2022-01-25 06:49:37--  https://github.com/opentrace-community/opentrace-ios/ar
Resolving github.com (github.com)... 13.237.44.5
Connecting to github.com (github.com)|13.237.44.5|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/uwacsp/opentrace/zip/refs/heads/master [fo
--2022-01-25 06:49:53--  https://codeload.github.com/uwacsp/opentrace/zip/refs/h
Resolving codeload.github.com (codeload.github.com)... 3.105.64.153
Connecting to codeload.github.com (codeload.github.com)|3.105.64.153|:443... con
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'master.zip'

master.zip                                                    [   <=>

2022-01-25 06:49:59 (6.46 MB/s) - 'master.zip' saved [3552764]
```

1. Unzip the file using the command `unzip`

3. Use the find command to search the `opentrace` directory for the file `AppDelegate.swift`

> ⓘ If you receive an error message "command not found", you can install the package yourself by typing in the shell:
>
> apt-get update
> apt-get install [package name]

Flag: Enter the directory that you found AppDelegate.swift in (from the root, the full path without the file and the trailing /) - it is case sensitive.

1.4. Processes

As this is Linux, we can also use the Linux command ps which together with the `-AF` flags shows all proesses and extended information

```
root@c31804846451:/cits1003/lab3/opentrace# ps -AF
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0  1062  3604   7 06:47 pts/0    00:00:00 bash
root       279     1  0  1475  2848   1 06:58 pts/0    00:00:00 ps -AF
```

In the output, the UID is the ID of the user that owns the process. On this container, that is `root`. The PID is the process ID and since this container is only running a bash shell, that is given the process ID of 1 with a parent process ID (PPID) of 0. The second process is the ps command we actually ran. This has a process ID of 279 (yours will likely be different) and a parent process ID of 1 as it was run from that bash shell.

We can run another `bash` shell and then try the ps command again:

```
root@c31804846451:/cits1003/lab3/opentrace# bash
root@c31804846451:/cits1003/lab3/opentrace# ps -AF
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0  1062  3604   7 06:47 pts/0    00:00:00 bash
root       280     1  0  1061  3656   4 06:59 pts/0    00:00:00 bash
root       283   280  0  1475  3028   6 06:59 pts/0    00:00:00 ps -AF
```

We can now see that there are the two `bash` processes listed. To show the hierarchy, we can use a program called `pstree`:

> package psmisc, so to install pstree , you must execute:
> apt-get install psmisc

```
root@c31804846451:/cits1003/lab3/opentrace# pstree -apG
bash,1
  ☼└─bash,280
      ☼└─pstree,319 -apG
```

So the parent `bash` process (1) is the parent, or root, of the tree, then the bash shell we ran (280) and finally the `pstree` command with the arguments `-apG` (show arguments, process ids and format the symbols).

Being able to show running processes is important when trying to detect processes running that shouldn't be. This is especially true in Windows where malware can disguise itself as a normal running process or even implant itself in a normal process.

To stop a process you can use the kill command with the process id. Sometimes, you can force the process to stop using the `-9` command:

```
root@c31804846451:/cits1003/lab3/opentrace# kill -9 280
Killed
root@c31804846451:/cits1003/lab3/opentrace# ps -AF
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0  0  1062  3604   0 06:47 pts/0    00:00:00 bash
root       320     1  0  1475  2884   4 07:03 pts/0    00:00:00 ps -AF
```

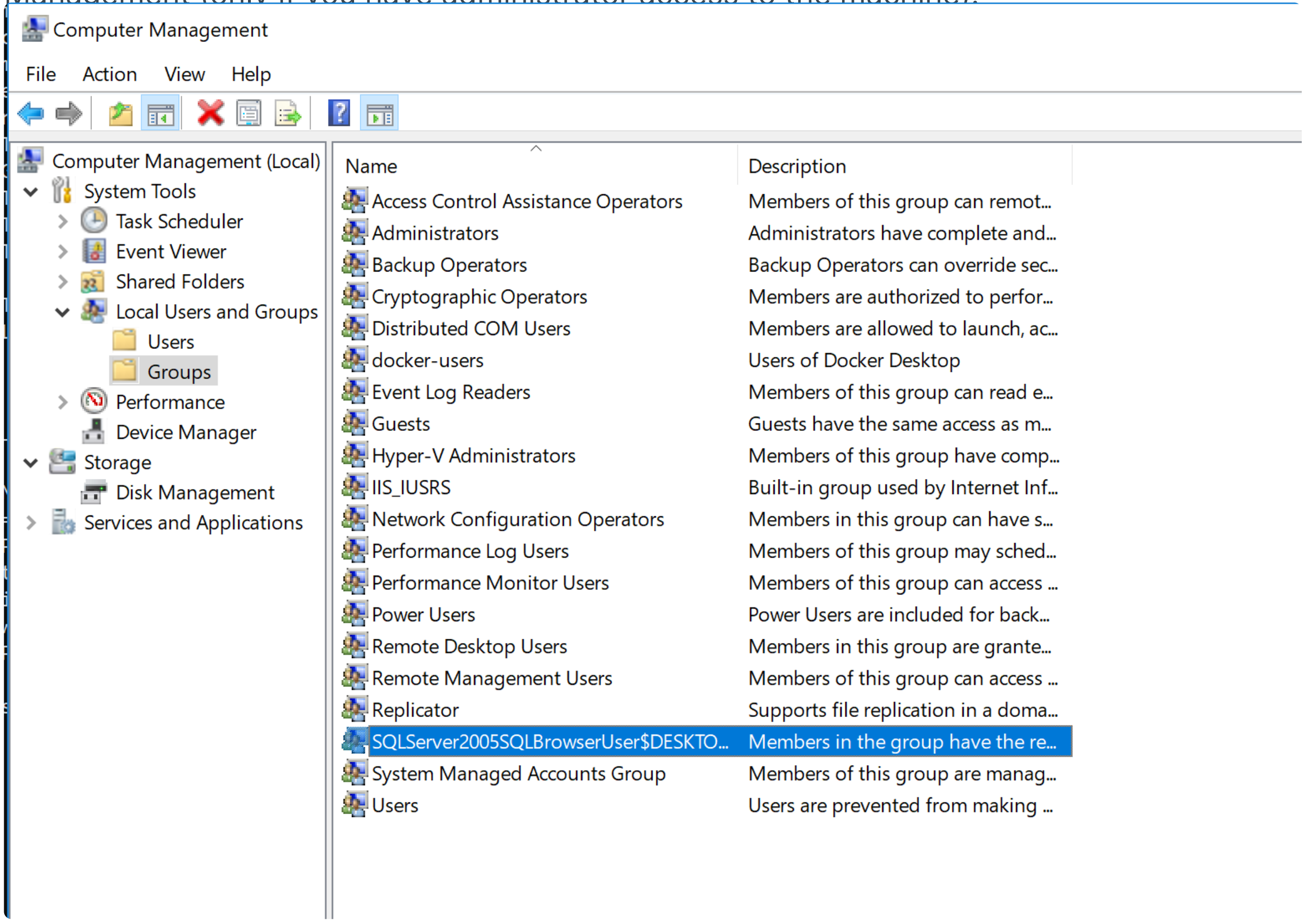
You can get detailed process information on Linux by looking at the file that this information is stored in. For example, if we wanted to get information about the bash process running in the container, we would use the command (replacing the <PID> with the process ID):

```
cat /proc/<PID>/status
```

1.5. Users and Groups

Both Windows and Linux implement Role Based Access Control (RBAC) based on groups of users. On Windows 10, as the principle user of a PC, you will likely have Administrator access and so you will be part of a group called `BUILTIN\Administrators`. Windows does not let you perform actions as part of this role however, so it will ask you for confirmation when you run an application as Administrator for example.

You can view details of the users and groups on your machine by looking at Computer Management (only if you have administrator access to the machine).



Name	Description
Access Control Assistance Operators	Members of this group can remot...
Administrators	Administrators have complete and...
Backup Operators	Backup Operators can override sec...
Cryptographic Operators	Members are authorized to perfor...
Distributed COM Users	Members are allowed to launch, ac...
docker-users	Users of Docker Desktop
Event Log Readers	Members of this group can read e...
Guests	Guests have the same access as m...
Hyper-V Administrators	Members of this group have comp...
IIS_IUSRS	Built-in group used by Internet Inf...
Network Configuration Operators	Members in this group can have s...
Performance Log Users	Members of this group may sched...
Performance Monitor Users	Members of this group can access ...
Power Users	Power Users are included for back...
Remote Desktop Users	Members in this group are grante...
Remote Management Users	Members of this group can access ...
Replicator	Supports file replication in a doma...
SQLServer2005SQLBrowserUser$DESKTO...	Members in the group have the re...
System Managed Accounts Group	Members of this group are manag...
Users	Users are prevented from making ...

Computer Management of Groups

There are a variety of ways of getting user and group information using PowerShell:

- Get-LocalUser will list the users of the machine
- Get-LocalGroup will list the groups
- Get-LocalGroupMember <group name> will list the members of a group

Again these won't work in PowerShell run on Linux

For Linux and Mac

Go back to the bash container you were running above. You can use `whoami` to get the currently logged in user:

```
root@c31804846451:~# whoami
root
```

id <user> will list the groups the user is a member of. Omitting <user> will look up your own groups.

```
root@c31804846451:~# id
uid=0(root) gid=0(root) groups=0(root)
```

If you want to know all of the groups on the computer, you can list the contents of the file `/etc/groups`

You can also list the passwd file which will list all of the users of the system

```
root@c31804846451:/cits1003/lab3/opentrace# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

1.6. File Permissions

Linux

All files on Linux have a user and a group that is assigned specific access to read (r), write (w) and execute (x) the file. Looking at the access control list of a file, you can see that the permissions are specified for the user, group and other. We can do this using the tool `getfacl`. Create a file called file.txt using the command `touch file.txt`. Then run

> `getfacl` is part of the package `acl`, i.e.,:
>
> `apt-get install acl`

```
root@c31804846451:/# getfacl file.txt
# file: file.txt
# owner: root
# group: root
user::rw-
```

We won't go into too much detail but this shows that the user has read and write access, the group root and everyone else has just read access. The file is not marked as being executable and so there is no 'x' involved.

You can also see the permissions using the `ls -al` command on Mac and Linux

```
root@c31804846451:/# ls -al file.txt
-rw-r--r-- 1 root root 0 Jan 25 07:11 file.txt
```

You can change permissions on files and directories using the `chmod` command. To mark a program as being executable for a user for example you can do:

```
root@c31804846451:/# chmod u+x file.txt
root@c31804846451:/# ls -al file.txt
-rwxr--r-- 1 root root 0 Jan 25 07:11 file.txt
```

Question 2. Run for the flag

Change directory into `/opt/lab3` and run the program `showflag` to get the flag. You will have to figure out why it won't run.

Flag: Enter the flag returned by showflag

> If you get an error running showflag on M1 Macs, please ask the facilitator for help.

Case study: Dirty Pipe

Dirty Pipe (CVE-2022-0847) proved that there is a new way to exploit Linux syscalls to write to files with a read-only privileges. The fact that someone can write to a file regardless of its permissions is a big security threat. An application of this vulnerability would be to write on the host from an unprivileged container. Keep in mind that this vulnerability is a kernel vulnerability which makes it hard, or even impossible, for user-mode runtime monitoring programs to detect this sort of file modification.

Read through the following article and answer the questions below: [https://blog.aquasec.com/deep-analysis-of-the-dirty-pipe-vulnerability]https://blog.aquasec.com/deep-analysis-of-the-dirty-pipe-vulnerability

Which of the following is true about Dirty Pipe (CVE-2022-0847)?

1. A Linux syscall to write to files with a read-only privilege.
2. A kernel vulnerability that allows writing to files with read-only privileges in Linux.
3. An application vulnerability that allows writing to files with read-only privileges in Linux.
4. A Linux kernel version released in 2022.
5. A way to copy files in Linux.

> Submit the correct option as your flag (e.g., `CITS1003{1}` if option 1 is the correct answer).

Question 4. sendfile syscall

What is the sendfile syscall used for?

1. To copy a file with a naive writing method.
2. To copy data between two files when at least one of them is a pipe.
3. To copy from one file to another without moving through the user-mode.
4. To issue data copy only when one modifies a copied data.

Question 5. Pipe in Linux

what is a pipe in Linux?

1. A ring buffer used for data transfer between two processes or threads.
2. A new vulnerability that allows writing to files with read-only privileges in Linux.
3. A system object which allows data transfer to and/or from them.
4. A new way to copy files in Linux.

←
CITS1003 Labs - Previous
Lab 2: Cryptography

Next - CITS1003 Labs