Universidad de Costa Rica

Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0117 – Programación Bajo Plataformas Abiertas
II ciclo 2025

Informe :

Laboratorio 2

Danny Garro Arias B93230

Profesora:
Carolina Trejos Quirós.

11 de setiembre de 2025

Índice

1. Introducción

El presente informe busca mostrar los resultados obtenidos, así como el procedimiento necesario para resolver los problemas de programación presentados. El primer problema consiste en la creación de un programa que reciba un nombre de usuario, un nombre de grupo y la ruta de un archivo, con el fin de conceder permisos especiales a dicho usuario y grupo. El segundo problema consiste en la elaboración de un script que muestre al usuario, en forma de gráficas, el consumo de cpu y memoria del proceso durante su ejecución.

2. Repositorio

En esta sección se brinda la dirección del repositorio en el cual se trabajó el presente laboratorio. La url del repositorio es la siguiente https://github.com/DG28828/plataformas_abiertas.git. Los archivos se encuentran ubicados en la siguiente dirección `/Laboratorios/2_Laboratorio`.

3. Scripting, usuarios y permisos

El objetivo de este ejercicio es crear un script que reciba un nombre de usuario, un nombre de grupo y la ruta de un archivo. El script debe hacer lo siguiente:

- Verificar que el usuario que ejecuta el script es *root*.
- Verificar que el usuario exista. Si no existe, crearlo.
- Verificar que el grupo exista. Si no existe, crearlo y asignar el usuario al grupo.
- Modificar la pertenencia del archivo al usuario indicado.
- Modificar los permisos al siguiente formato (- *r w x r* - - - - -).

Se comenzó declarando las variables *usuario*, *grupo* y *ruta*, correspondientes a las entradas argumentales *$1*, *$2* y *$3*, respectivamente. Luego de esto, se emplearon instrucciones para la creación de un archivo para registrar los mensajes relevantes durante la ejecución del script, llamado `registro_ejecucion.log`. En cada ejecución del script se elimina el archivo log y se crea uno nuevo, cuyo encabezado contiene la fecha del momento de la ejecución.

```
#!/bin/bash

usuario=$1
grupo=$2
ruta=$3

if [[ -e registro_ejecucion.log ]]; then
    rm registro_ejecucion.log
fi
touch registro_ejecucion.log
echo "## Registro de ejecución del script ##" >> registro_ejecucion.log
echo "Fecha de creación: `date`" >> registro_ejecucion.log
echo " " >> registro_ejecucion.log
```

La salida del archivo log es de la siguiente forma:

```
$ cat registro_ejecucion.log

## Registro de ejecución del script ##
Fecha de creación: Wed Sep 10 16:18:24 CST 2025
```

Luego de esto se crearon instrucciones para verificar si el usuario que ejecuta el script es *root*. Esto se logró verificando el contenido de la variable de entorno *$EUID*, cuyo valor es igual a cero cuando el usuario que ejecuta es root, y es distinto de cero en caso contrario. Por lo tanto, el requerimiento se puede satisfacer al verificar el contenido de dicha variable de entorno de la siguiente forma:

```
if [[ $EUID -ne 0 ]]; then
    echo "ERROR: El usuario actual no es el usuario root. Finalizando ejecución"  | tee
    -a registro_ejecucion.log
    exit 2
fi
```

Nótese que se empleó un *pipe* para redireccionar el mensaje de error al comando **tee** con la opción *-a* cuya función en el presente caso es dirigir la salida estándar de **echo** a una escritura en el archivo log y a la terminal de forma simultánea. La salida cuando el usuario que ejecuta no es *root* se ve de la siguiente forma:

```
$ ./ejercicio1.sh danny danny prueba.txt

ERROR: El usuario actual no es el usuario root. Finalizando ejecución
```

y el archivo de log:

```
$ cat registro_ejecucion.log

## Registro de ejecución del script ##
Fecha de creación: Wed Sep 10 16:48:05 CST 2025

ERROR: El usuario actual no es el usuario root. Finalizando ejecución
```

Para los siguientes ejemplos se mostrará únicamente los resultados de la salida en terminal, pero puede verificarse que los mensajes se van almacenando en el archivo de log.

Lo siguiente fue verificar la existencia del archivo indicado en la ruta; para esto se emplea el comando *-e* seguido de la ruta.

```
if ! [[ -e $ruta ]]; then
    echo "ERROR: La ruta no existe. Finalizando ejecución" | tee -a registro_ejecucion.log
    exit 2
fi
```

lo cual brinda la siguiente salida si el archivo no existe:

```
$ ./ejercicio1.sh danny danny prueba1.txt

ERROR: La ruta no existe. Finalizando ejecución
```

Lo siguiente es verificar la existencia grupo indicado, para lo cual se emplea el comando **getent**, cuya función es recuperar entradas de la base de datos administrativa del sistema. Al escribir *group* y el nombre del grupo, tras escribir el comando, se obtiene lo siguiente: una cadena de texto de información del grupo si este existe, o una salida vacía si este no existe. Por lo tanto, basta con verificar que la salida del comando sea vacía para comprobar que el grupo no existe. Si el grupo no existe, este se crea con el comando **addgroup**, seguido del nombre del grupo. A continuación se muestra el código corespondiente:

```
verificar_grupo=`getent group $grupo`
if [[ $verificar_grupo = "" ]]; then
    echo "El grupo $grupo no existe." | tee -a registro_ejecucion.log
    echo "Creando grupo" | tee -a registro_ejecucion.log
    addgroup $grupo | tee -a registro_ejecucion.log
else
    echo "El grupo $grupo existe. Información del grupo: $verificar_grupo"  | tee -a
    registro_ejecucion.log
fi
```

Esto brinda los siguientes resultados: Si el grupo no existe:

```
$ ./ejercicio1.sh danny grupo1 prueba.txt

El grupo grupo1 no existe.
Creando grupo
info: Selecting GID from range 1000 to 59999 ...
info: Adding group `grupo1' (GID 1001) ...
```

En cambio, si el grupo existe (se prueba con el grupo que se acaba de crear):

```
$ ./ejercicio1.sh danny grupo1 prueba.txt

El grupo grupo1 existe. Información del grupo: grupo1:x:1001:danny
```

Para verificar la existencia del usuario se hace exactamente el mismo procedimiento, pero ahora indicando *passwd*, seguido del nombre de usuario. Si el usuario no existe, este se crea con el comando **adduser**.

```
verificar_usuario=`getent passwd $usuario`
if [[ $verificar_usuario = "" ]]; then
    echo "El usuario $usuario no existe." | tee -a registro_ejecucion.log
    echo "Creando usuario" | tee -a registro_ejecucion.log
    adduser $usuario | tee -a registro_ejecucion.log
else
    echo "El usuario $usuario existe. Información del usuario: $verificar_usuario"  | tee
    -a registro_ejecucion.log
fi
```

Si el usuario no existe, se obtiene la siguiente salida:

```
$ ./ejercicio1.sh nuevo_usuario danny prueba.txt

El grupo danny existe. Información del grupo: danny:x:1000:danny
El usuario nuevo_usuario no existe.
Creando usuario
info: Adding user `nuevo_usuario' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `nuevo_usuario' (1002) ...
info: Adding new user `nuevo_usuario' (1002) with group `nuevo_usuario (1002)' ...
info: Creating home directory `/home/nuevo_usuario' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for nuevo_usuario
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:  Home Phone []:

        Other []:

Is the information correct? [Y/n]
```

Si el usuario existe, se obtiene lo siguiente:

```
$ ./ejercicio1.sh nuevo_usuario danny prueba.txt

El grupo danny existe. Información del grupo: danny:x:1000:danny
El usuario nuevo_usuario existe. Información del usuario: nuevo_usuario:x:1002:1002:,
,,:/home/nuevo_usuario:/bin/bash
```

Hasta este punto, siempre se tendrá creado el usuario y el grupo indicados; lo siguiente corresponde a asignar el usuario al grupo. Para esto se emplea el comando **usermod**, seguido de las opciones *-a* y *-G*, y seguido del nombre de usuario y el grupo.

```
usermod -a -G $grupo $usuario  | tee -a registro_ejecucion.log
```

Se puede verificar que se agregó satisfactoriamente al nuevo grupo con el comando **groups**, desde el usuario nuevo. Al ejecutar los comandos que se muestran a continuación (no se muestra la salida desglosada de ejecutar el script), se ve que el nuevo usuario pertenece a un grupo con su nombre (que se creó predeterminadamente durante la creación del usuario) y al nuevo grupo.

```
$ ./ejercicio1.sh nuevo_usuario2 nuevo_grupo2 prueba.txt
$ su nuevousuario2
```

```
$ groups

nuevo_usuario2 nuevo_grupo2
```

Se procede a modificar la pertenencia del archivo, para lo cual se emplea el comando **chown**, seguido de *$usuario:$grupo $ruta*. Finalmente, se modifican los permisos del archivo. Los permisos requeridos corresponden al número octal 740, por lo tanto:

```
echo "Modificando pertenencia del archivo $ruta al usuario $usuario y el grupo $grupo"
chown $usuario:$grupo $ruta
echo "Modificando permisos a la siguiente configuración:"
echo "    Usuario: Lectura, Escritura y Ejecución"
echo "    Grupo: Lectura"
echo "    Otros: -ninguno-"
chmod 740 $ruta
```

Antes de mostrar el funcionamiento, se muestra el estado actual de los archivos del directorio al eliminar y crear nuevamente el archivo `prueba.txt`.

```
$ ls -l

total 12
-rwxr-xr-x 1 danny danny 2170 Sep 10 19:15 ejercicio1.sh
-rw-r--r-- 1 danny danny   13 Sep  8 15:29 ejercicio1.sh~
-rw-r--r-- 1 root  root     0 Sep 10 19:34 prueba.txt
-rw-r--r-- 1 root  root   288 Sep 10 19:33 registro_ejecucion.log
```

Al ejecutar el script:

```
$ ./ejercicio1.sh nuevo_usuario2 nuevo_grupo2 prueba.txt
$ ls -l

total 12
-rwxr-xr-x 1 danny          danny         2170 Sep 10 19:15 ejercicio1.sh
-rw-r--r-- 1 danny          danny           13 Sep  8 15:29 ejercicio1.sh~
-rwxr----- 1 nuevo_usuario2 nuevo_grupo2     0 Sep 10 19:34 prueba.txt
-rw-r--r-- 1 root           root          302 Sep 10 19:38 registro_ejecucion.log
```

Con lo cual se muestra que ahora el archivo `prueba.txt` pertenece *nuevo_usuario2* y al grupo *nuevo_grupo2*. Además, los permisos cambiaron a (- *r w x r* - - - - -).

cat

4. Scripting y procesos

EL objetivo de este ejercicio es crear una herramienta que permita monitorear el consumo de CPU y memoria de un proceso recibido como argumento. AL finalizar el proceso, el script

debe mostrar gráficas del consumo durante su ejecución.

Para comenzar, el script crea dos archivos de log `consumo_cpu.log` y `consumo_mem.log`, donde se registrará el consumo de CPU y memoria del proceso, respectivamente. La creación de estos archivos se realizó con las mismas instrucciones que se mostraron en la creación del archivo de log del primer ejercicio. Seguido de esto, se declaró una variable llamada *proceso* en la que se guarda el nombre del proceso a ejecutar y posteriormente se ejecuta el proceso de la siguiente forma:

```
$proceso &
PID=$!
```

Lo anterior ejecuta el proceso y guarda el identificador (PID) en la variable *PID*.

Seguido, se implementa un ciclo while, en el que se verifica la existencia del directorio asignado al proceso ejecutado dentro de la ruta `/proc`. Mientras el proceso exista, el script guarda en los respectivos archivos la hora y el consumo. Para lo anterior se implementó el comando **ps**, seguido de la opción *-p*, que permite filtrar los procesos por su PID; luego la opción *-o*, que formatea la salida a lo especificado por el usuario, en este caso interesa *%cpu* Y *%mem*; finalmente, se utiliza la opción *–no-headers* para que se imprima únicamente el valor especificado y no su encabezado. Con el comando **sleep**, seguido de un 1, se logra que los datos se muestreen cada segundo.

```
while [[ -d /proc/$PID ]]; do
    echo "`date +%T` `ps -p $PID -o %cpu --no-headers`" >> consumo_cpu.log
    echo "`date +%T` `ps -p $PID -o %mem --no-headers`" >> consumo_mem.log
    sleep 1
done
```

AL correr el script con el proceso *chromium* y verificar la salida de los archivos, se obtienen las siguientes salidas:

```
$ ./ejercicio2. sh chromium

$ cat consumo_cpu.log

21:02:50  0.0
21:02:51 44.0
21:02:52 22.0
21:02:53 19.6
21:02:54 14.7
21:02:55 12.0
21:02:56 11.0
21:02:57  9.4
21:02:59  7.3
21:03:00  6.6
```

```
21:03:01  6.0
21:03:02  5.5
21:03:03  5.1
21:03:04  5.2

$ cat consumo_mem.log

21:02:50  0.0
21:02:51  0.9
21:02:52  0.9
21:02:53  1.0
21:02:54  1.0
21:02:55  1.0
21:02:56  1.0
21:02:58  1.0
21:02:59  1.0
21:03:00  1.0
21:03:01  1.0
21:03:02  1.0
21:03:03  1.0
21:03:04  1.0
```

Finalmente, para graficar los valores, se emplea la funcionalidad **gnuplot**, que recibe como entrada los datos de las columnas de los archivos indicados, y tras formatear adecuadamente, se obtienen los resultados que se muestran en las Figuras 1 y 2.

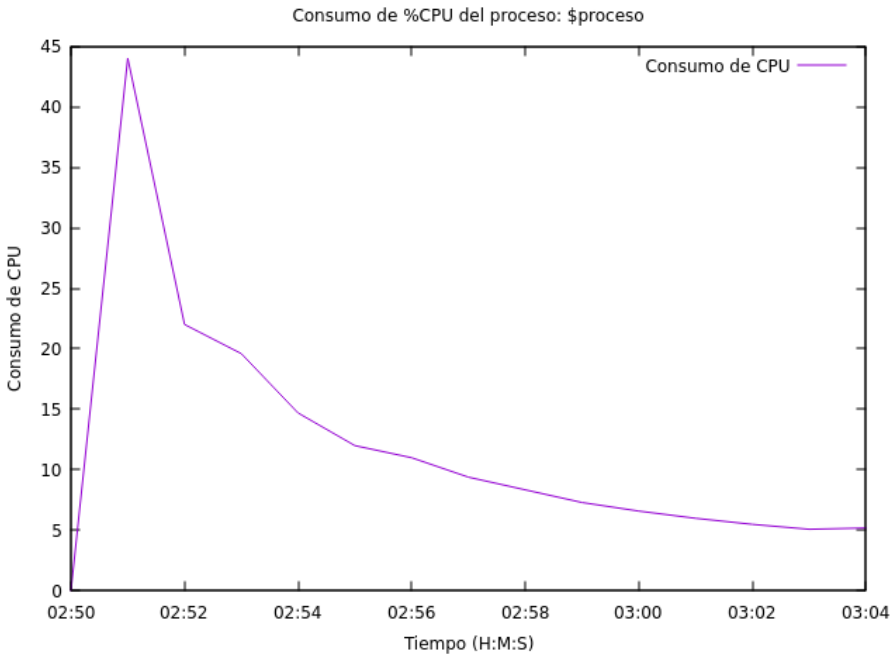

Figura 1: Consumo de CPU del proceso

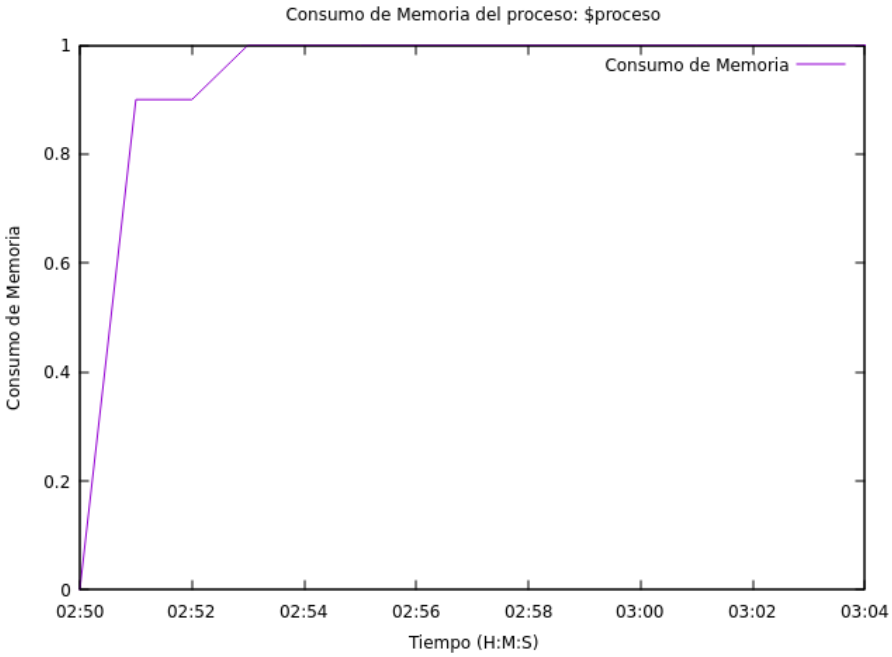

Figura 2: Consumo de memoria del proceso