UNIVERSIDAD DE COSTA RICA

Escuela de Ingeniería Eléctrica

IE0117: Programación Bajo Plataformas Abiertas

---

# Reporte

## Laboratorio #2

---

Prof.Carolina Trejos Quiros

Estudiante: Jafet Guillermo Cruz Salazar - C02520

30 de marzo de 2025

# Índice

# 1. Introducción

El presente laboratorio tuvo como objetivo principal la aplicación práctica de comandos y herramientas del sistema operativo GNU/Linux para el desarrollo de scripts de automatización, gestión de procesos y monitoreo del sistema. A lo largo de este laboratorio se abordaron tareas fundamentales de administración en `bash`[1], tales como el cambio de permisos y propietarios con comandos como `chmod`, `chown`, y la gestión de usuarios y grupos mediante `useradd` y `groupadd`[2]. También se abordó la captura de métricas de rendimiento con herramientas como `ps`[2] y la creación de servicios personalizados usando `systemd`[3].

Además, se exploró el uso de utilidades como `inotifywait`[4] para el monitoreo de eventos en directorios y `gnuplot`[5] para la representación visual de datos de monitoreo. Se diseñaron scripts capaces de adaptarse a distintos entornos, como WSL (Windows Subsystem for Linux) y distribuciones Linux nativas.

Este laboratorio también sirvió como una introducción práctica a la configuración de servicios de usuario, la integración de registros (`.log`) y la representación gráfica del rendimiento del sistema.

El código fuente de los scripts desarrollados se encuentra disponible en el siguiente repositorio, en dicho repositorio también se encuentran instrucciones para la ejecución de cada script : https://github.com/MavrosAilouros/Lab2_Administration_permissions

# 2. Implementación

## Ejercicio 1: Gestión de usuarios, grupos y permisos

**Objetivo:** Diseñar un script capaz de gestionar de forma automatizada la propiedad y permisos de un archivo en el sistema. El script debe verificar o crear un usuario y un grupo, asociarlos correctamente, validar que el archivo pertenezca a ambos, y asegurar que tenga permisos `740`.

**Desarrollo:** El ejercicio fue abordado mediante un enfoque secuencial, priorizando la verificación del entorno antes de aplicar cualquier modificación. Se establecieron condiciones de seguridad, como la exigencia de ejecutar el script como `root`, y la validación estricta de los parámetros.

El script recibe tres argumentos: usuario, grupo y ruta del archivo. A partir de allí, se diseñaron los siguientes pasos:

1. Validar que el script se ejecute como `root` y con tres parámetros exactos.
2. Verificar si el grupo existe; si no, crearlo.
3. Verificar si el usuario existe; si no, crearlo asociado al grupo.
4. En caso de que el usuario exista pero no pertenezca al grupo, asociarlo.
5. Verificar si el archivo existe.
6. Si el archivo no es propiedad del usuario y grupo indicados, cambiar la propiedad.
7. Verificar los permisos del archivo y asignar
8. Registrar toda la ejecución en un archivo de log llamado `ejercicio1.log`. `740` si es necesario.

El siguiente fragmento ilustra cómo se valida si el script es ejecutado como `root`, condición necesaria para modificar usuarios y grupos:

Listing 1: Validación como root

```
if [ "$EUID" -ne 0 ]; then
    echo "Este script debe ser ejecutado como Root" | tee -a "$log"
    exit 1
fi
```

Para la gestión de grupos, se utilizó el comando `getent` que permite verificar la existencia sin imprimir errores innecesarios:

Listing 2: Verificación y creación de grupo

```
if getent group "$Group" > /dev/null; then
    echo "El grupo $Group ya existe." | tee -a "$log"
else
    echo "El grupo $Group no existe. Creando grupo..." | tee -a "$log"
    groupadd "$Group"
fi
```

Posteriormente se manejó la existencia del usuario, su asociación al grupo, y la creación completa si era necesario. Se optó por una estructura que evita sobreescribir configuraciones existentes y muestra mensajes claros para facilitar el análisis:

Listing 3: Verificación o creación del usuario y asignación al grupo

```
if id "$User" &>/dev/null; then
    echo "El usuario $User ya existe."
    if id -nG "$User" | grep -qw "$Group"; then
        echo "El usuario $User ya pertenece al grupo $Group." |tee -a "$log"
    else
        usermod -aG "$Group" "$User"
    fi
else
    useradd -m -g "$Group" "$User"
fi
```

Finalmente, se gestionó la propiedad y permisos del archivo únicamente si eran diferentes de lo esperado. Esto se hizo para reducir la cantidad de comandos innecesarios ejecutados por el sistema:

Listing 4: Cambio condicional de propiedad y permisos

```
read owner group <<< $(stat -c "%U %G" "$File")
if [ "$owner" != "$User" ] || [ "$group" != "$Group" ]; then
  chown "$User:$Group" "$File"
fi

perms=$(stat -c "%a" "$File")
if [ "$perms" != "740" ]; then
  chmod 740 "$File"
fi
```

Este ejercicio integró múltiples aspectos del sistema Linux relacionados con la administración de usuarios y control de acceso. El uso de las condiciones diseñadas permitió construir un script simple, reutilizable y seguro. Fue importante comprender la interacción entre usuarios, grupos y permisos para evitar errores o conflictos con configuraciones existentes.

## Ejercicio 2: Monitoreo del uso de recursos del sistema

**Objetivo:** El segundo ejercicio consistió en desarrollar un script llamado `ejercicio2.sh` capaz de ejecutar un programa dado como argumento, monitorear periódicamente el uso de CPU y memoria durante su ejecución, registrar estos datos en un archivo `.log`, y graficar automáticamente los resultados usando `gnuplot`.

**Desarrollo:** El script se diseñó para aceptar el nombre de cualquier proceso o comando como argumento, lo que permite monitorear fácilmente distintas aplicaciones. Es importante señalar que el proceso debe ser cerrado manualmente para procesos que no tienen un tiempo finito de ejecución (como Firefox). De lo contrario, el script continuará ejecutándose indefinidamente.

Además, durante la ejecución del script en un sistema Linux nativo se detectó un comportamiento particular en procesos gráficos como Firefox, donde el proceso queda abierto luego de ejecutar el script y no se cierra automáticamente. Esto debe ser tomado en cuenta por el usuario, quien debe cerrar manualmente el proceso monitoreado para detener el script y generar el gráfico correctamente.

**Observación:** En algunos casos se observó que el valor de uso de CPU superó el 100 % (por ejemplo, 104 %). Esto se debe a que en sistemas con múltiples núcleos, el comando `ps` reporta el uso de CPU en relación con un solo núcleo. Por lo tanto, si un proceso utiliza más de un núcleo parcialmente o completamente, el porcentaje total puede exceder el 100 %. Este comportamiento es normal y refleja un uso combinado de varios núcleos.

A continuación, se describen los pasos implementados y las decisiones tomadas durante el proceso.

1. Ejecucion del proceso objetivo en segundo plano mediante `setsid`, para mantener un enlace de sesión robusto y obtener su PID.

2. Monitoreo continuo del proceso usando el comando `ps`, registrando cada segundo el consumo de CPU y memoria en un archivo log.

3. Una vez finalizado el proceso, generación automática de un gráfico con los resultados obtenidos utilizando la herramienta `gnuplot`.

El proceso objetivo se lanza en segundo plano, capturando su identificador (PID) para control preciso:

Listing 5: Lanzamiento del proceso y captura del PID

```
setsid $command &
pid=$!
echo "Ejecutando "$command", PID: $pid"
```

Luego, se genera un archivo de registro inicial con encabezados claros para facilitar la posterior generación de gráficos:

Listing 6: Creación del archivo de registro

```
log_file="reporte_de_uso.log"
printf "%-8s %-7s %-7s\n" "Segundos" "CPU(%)" "MEM(%)" > $log_file
```

Para monitorear el uso de CPU y memoria, se emplea un bucle condicional que verifica cada segundo la existencia del proceso y registra los datos obtenidos mediante `ps`:

Listing 7: Bucle de monitoreo y registro de datos

```
SECONDS_SINCE_START=0
```

```
while kill -0 "$pid" 2>/dev/null; do
    Usage=$(ps -p $pid -o %cpu,%mem --no-headers)
    printf "%-8s %-7s %-7s\n" "$SECONDS_SINCE_START" $Usage >> $log_file
    sleep 1
    SECONDS_SINCE_START=$((SECONDS_SINCE_START + 1))
done
```

El uso del comando `kill -0` permitió determinar eficientemente la existencia del proceso sin interrumpirlo ni afectarlo negativamente.

Finalmente, se genera automáticamente un gráfico claro utilizando `gnuplot` con la configuración específica siguiente:

Listing 8: Configuración final del gráfico con gnuplot

```
set terminal png size 800,600
set output "grafico_uso.png"
set title "Consumo de CPU y Memoria"
set xlabel "Segundos"
set ylabel "Uso (%)"
set yrange [0:*]
set grid
plot "reporte_de_uso.log" using 1:2 with lines title "CPU (%)", \
     "reporte_de_uso.log" using 1:3 with lines title "Memoria (%)"
```

Este script automatiza la tarea esencial del monitoreo de recursos del sistema, generando datos valiosos y visualmente interpretables para evaluar el rendimiento de cualquier aplicación. Su diseño facilita el monitoreo repetitivo, lo que representa una herramienta útil para administradores y desarrolladores.

## Ejercicio 3: Monitoreo de cambios en directorios mediante servicios personalizados

**Objetivo:** Este ejercicio se enfocó en desarrollar un script llamado `ejercicio3.sh` que monitoree continuamente un directorio específico, detectando y registrando eventos como creación, modificación y eliminación de archivos. Además, se creó un servicio personalizado utilizando `systemd` para ejecutar este script automáticamente en segundo plano.

**Desarrollo:** Para resolver este ejercicio, se decidió utilizar la herramienta `inotifywait`, que permite monitorear eventos en directorios de forma eficiente y precisa. La implementación consideró especialmente la compatibilidad con WSL (Windows Subsystem for Linux), tomando decisiones que evitaran conflictos con las limitaciones particulares de este entorno.

Los pasos específicos seguidos fueron:

1. Validar que el script reciba un único argumento (el directorio a monitorear).
2. Verificar la existencia del directorio especificado.
3. Registrar eventos de creación, modificación y eliminación con fecha y hora en un archivo de log persistente.
4. Crear un servicio personalizado con `systemd` para ejecutar automáticamente el script en cada sesión del usuario.

La validación inicial garantizó la correcta ejecución del script:

Listing 9: Validación de argumentos

```
if [ "$#" -ne 1 ]; then
    echo "Uso: $0 <directorio_a_monitorear>"
    exit 1
fi
```

Para manejar correctamente la ubicación del log en un entorno como WSL, se optó por almacenar el archivo log en el directorio personal del usuario, garantizando persistencia y facilidad de acceso:

Listing 10: Establecimiento del archivo de log

```
directory="$1"
log_file="$HOME/monitor.log"  # Archivo de log persistente por usuario
```

El monitoreo continuo del directorio seleccionado se realizó mediante un bucle infinito y la herramienta `inotifywait`, configurada para registrar claramente cada evento con la fecha y hora precisas:

Listing 11: Monitoreo continuo con inotifywait

```
while true; do
  event=$(inotifywait -e create -e modify -e delete \
          --format '%T %w %e %f' \
          --timefmt '%F %T' "$directory" 2>/dev/null)
  echo "Evento detectado: $event"
  echo "$event" >> "$log_file"
done
```

Para la ejecución automática y persistente del script, se creó un servicio personalizado de usuario en `systemd`. El archivo de configuración funcional resultante es el siguiente:

Listing 12: Archivo .service funcional

```
[Unit]
Description=Servicio de Monitoreo de Cambios en Directorios
After=network.target

[Service]
Type=simple
ExecStart=/bin/bash /ruta/completa/al/ejercicio3.sh /ruta/del/directorio
Restart=always

[Install]
WantedBy=default.target
```

Se tuvo especial cuidado con WSL, ya que el uso de rutas dinámicas mediante plantillas (`@.service`) presentó complicaciones debido a la necesidad de escapar caracteres especiales como las barras inclinadas (`/`). Por lo tanto, se decidió utilizar rutas absolutas fijas en este entorno, documentando claramente en el reporte la limitación encontrada y cómo manejarla correctamente.

**Problemas encontrados con WSL:** Durante el desarrollo se encontró que WSL requiere ajustes adicionales para usar unidades plantillas (`@.service`), debido a las dificultades relacionadas con rutas escapadas, especialmente cuando estas incluyen múltiples directorios anidados. Esto obligó a simplificar el diseño final, optando por una ruta absoluta fija en el archivo `.service` para asegurar un funcionamiento estable.

Este ejercicio evidenció la importancia de conocer las particularidades del entorno de trabajo, especialmente con plataformas híbridas como WSL. Las decisiones tomadas fueron fundamentales para asegurar que el script y el servicio resultaran robustos, funcionales y fáciles de utilizar.

# 3. Resultados

## Ejercicio 1: Gestión de usuarios, grupos y permisos

A continuación se presentan los resultados obtenidos al ejecutar el script `ejercicio1.sh` en diferentes escenarios:

- Caso 1: El archivo existe, pero su propietario y permisos no corresponden al usuario y grupo deseados.
- Caso 2: El archivo ya tiene la propiedad y permisos correctos.
- Caso 3: El usuario no pertenece al grupo especificado.

```
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$ sudo bash ejercicio1.sh mavros_ailour
os grupo1 /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt
El grupo grupo1 no existe. Creando grupo...
El usuario mavros_ailouros ya existe.
Agregando el usuario mavros_ailouros al grupo grupo1...
El usuario mavros_ailouros ha sido agregado al grupo grupo1.
El archivo /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt existe.
Cambiando la propiedad del archivo a mavros_ailouros:grupo1...
Cambiando permisos a 740...
Permisos cambiados a 740.
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$
```

Figura 1: Ejecución del script donde se corrige la propiedad y los permisos del archivo.

```
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$ sudo bash ejercicio1.sh mavros_ailour
os grupo1 /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt
El grupo grupo1 ya existe.
El usuario mavros_ailouros ya existe.
El usuario mavros_ailouros ya pertenece al grupo grupo1.
El archivo /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt existe.
El archivo ya es propiedad de mavros_ailouros:grupo1.
El archivo ya tiene permisos 740.
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$
```

Figura 2: Ejecución del script donde el archivo ya cuenta con la propiedad y permisos adecuados.

```
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$ sudo bash ejercicio1.sh mavros_ailour
os grupo2 /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt
[sudo] password for mavros_ailouros:
El grupo grupo2 no existe. Creando grupo...
El usuario mavros_ailouros ya existe.
Agregando el usuario mavros_ailouros al grupo grupo2...
El usuario mavros_ailouros ha sido agregado al grupo grupo2.
El archivo /home/mavros_ailouros/Universidad/Clases/IE-0117/Laboratorio2/prueba.txt existe.
Cambiando la propiedad del archivo a mavros_ailouros:grupo2...
El archivo ya tiene permisos 740.
mavros_ailouros@DESKTOP-44DT8L4:~/Universidad/Clases/IE-0117/Laboratorio2$
```

Figura 3: Ejecución del script con grupo inexistente.

## Ejercicio 2: Monitoreo de uso de CPU y memoria

El siguiente conjunto de capturas muestra la ejecución del script `ejercicio2.sh`, el archivo de log generado y el gráfico de uso de recursos. En este caso, se monitoreó un proceso activo por varios segundos.

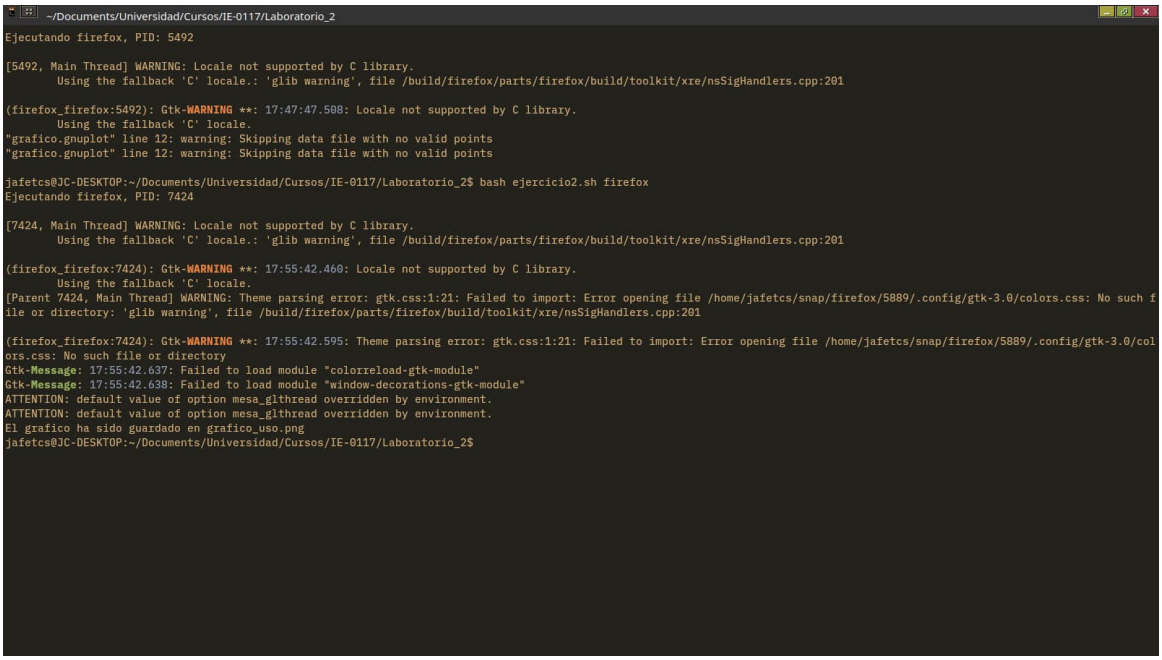


Figura 4: Ejecución del script con el monitoreo activo de un proceso.

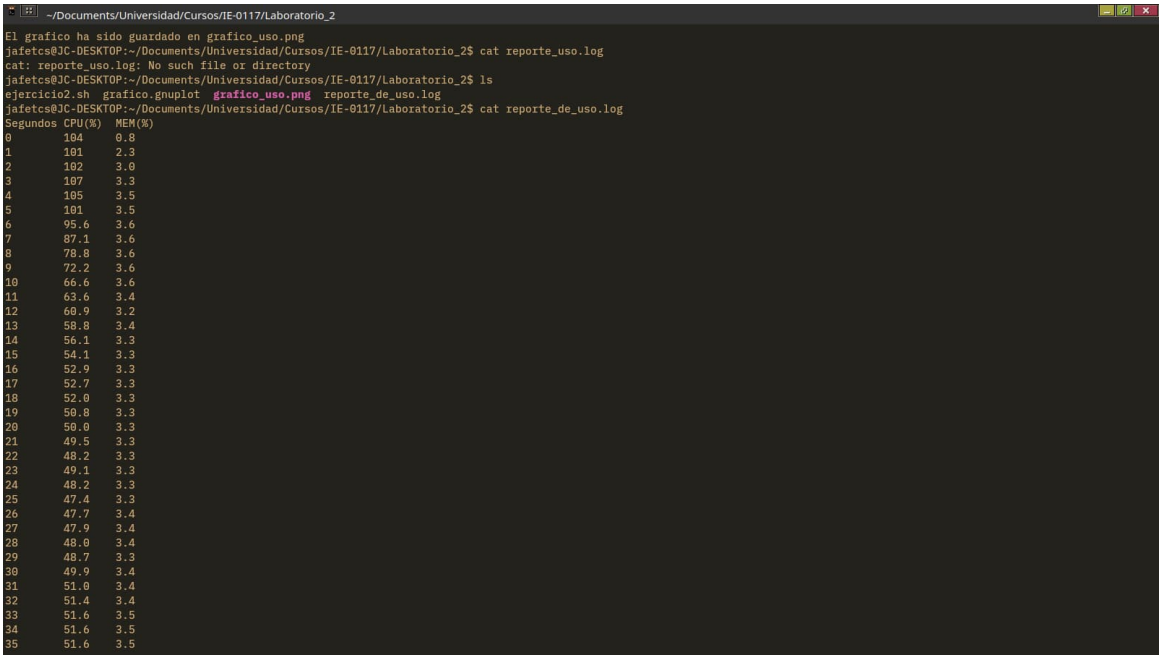


Figura 5: Contenido del archivo `reporte_de_uso.log` generado durante el monitoreo.

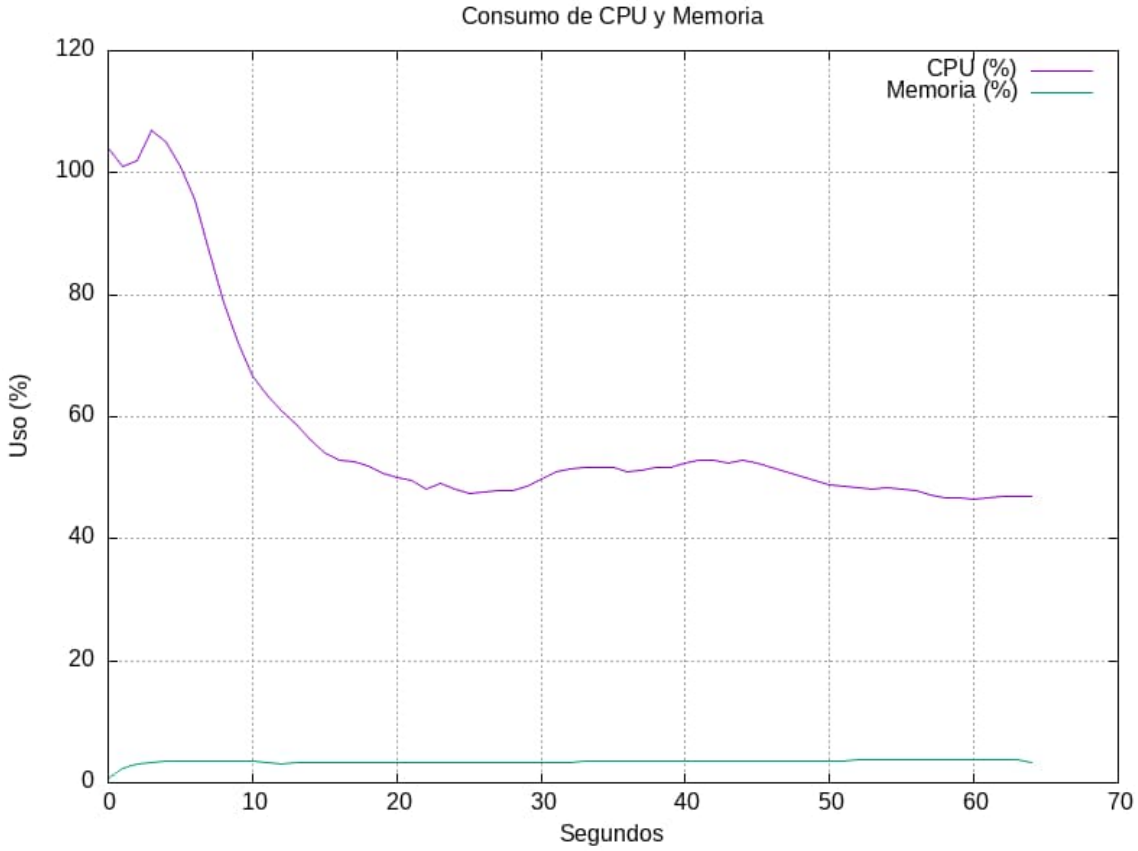


Figura 6: Gráfico generado por Gnuplot con el uso de CPU y memoria a lo largo del tiempo.

## Ejercicio 3: Monitoreo de eventos en directorios con un servicio personalizado

A continuación se presentan los resultados obtenidos al ejecutar el servicio `monitoreo-cambios.service` y generar eventos en el directorio monitoreado.


Figura 7: Estado del servicio activo utilizando `systemctl -user status`.


Figura 8: Eventos generados al crear, modificar y eliminar archivos en el directorio monitoreado.

# 4. Conclusiones y Recomendaciones

**Conclusiones**

El desarrollo de este laboratorio permitió aplicar conceptos fundamentales de administración en sistemas GNU/Linux mediante el uso de scripts en Bash. A lo largo de los tres ejercicios, se reforzó el conocimiento sobre:

- Manejo de permisos, usuarios y grupos, aplicando comandos como `chmod`, `chown`, `useradd`, y `groupadd`.
- Monitoreo del uso de recursos de procesos en tiempo real, integrando herramientas como `ps`, `bc`, y `gnuplot` para registrar y visualizar el comportamiento del sistema.
- Configuración y ejecución de servicios personalizados usando `systemd`, con monitoreo activo de directorios mediante `inotifywait`.

También se identificaron desafíos técnicos importantes, como el manejo de rutas en archivos `.service` en entornos como WSL, y la interpretación del uso del CPU mayor al 100 % en sistemas multinúcleo. Estos aspectos requirieron una comprensión más profunda del funcionamiento del sistema operativo y de los procesos.

**Recomendaciones**

- Es recomendable validar cuidadosamente los argumentos de entrada en los scripts, especialmente si se trabaja con usuarios, grupos o rutas sensibles del sistema.
- En entornos WSL, evitar el uso de plantillas dinámicas en archivos `.service` y optar por rutas absolutas bien definidas.
- Para el monitoreo de procesos, elegir comandos que no se desvinculen inmediatamente (como `sleep` o `gedit`) cuando se busca capturar su comportamiento en tiempo real.
- Documentar cada paso del script con mensajes claros al usuario y mantener un formato consistente, mejora la mantenibilidad y comprensión del código.
- Normalizar valores como el uso de CPU para mejorar la interpretación de los resultados, especialmente al trabajar con múltiples núcleos.

# Referencias

1. GNU Project, *GNU Bash Manual*, Marzo 2025. Available at https://www.gnu.org/software/bash/manual/.

2. GNU Project, *GNU Core Utilities Manual*, Marzo 2025. Available at https://www.gnu.org/software/coreutils/manual/.

3. Freedesktop.org, *systemd.service - systemd Service Manual*, Marzo 2025. Available at https://www.freedesktop.org/software/systemd/man/systemd.service.html.

4. L. man-pages project, *inotifywait - man page*, Marzo 2025. Available at https://man7.org/linux/man-pages/man1/inotifywait.1.html.

5. gnuplot team, *gnuplot Documentation*, Marzo 2025. Available at http://www.gnuplot.info/documentation.html.