

Laboratorio 2: Monitoreo y Procesamiento de Procesos del Sistema con Bash y Pipes

Curso de Sistemas Operativos

Segundo semestre 2025

Objetivos generales

Este laboratorio busca que las y los estudiantes implementen llamados al sistema:

1. Utilizar `fork()`, para crear procesos.
2. Hacer uso de `pipes()`, para la comunicación entre procesos.
3. Emplear `exec()`, para ejecutar procesos.
4. Duplicar descriptores con `dup2()`.

Objetivos del aprendizaje

Al finalizar el laboratorio, las y los estudiantes serán capaces de:

1. Capturar y validar parámetros por medio de `getopt` en lenguaje C.
2. Compilación por partes de programas en C, mediante un archivo `Makefile`.
3. Crear procesos mediante `fork()`, y establecer comunicación por `stdin/stdout`.
4. Mediante llamadas al SO, construir un pipeline que trabaje con scripts de `bash`.

Contexto y Motivación

Los sistemas operativos basados en Linux constituyen la infraestructura dominante de la computación moderna: impulsan la mayoría de servidores en Internet, gran parte de la nube pública y privada, dispositivos embebidos, supercomputación y contenedores. Su relevancia no solo radica en el rendimiento y la estabilidad del kernel, sino también en su ecosistema POSIX y en la filosofía Unix de componer soluciones con herramientas pequeñas, reutilizables y observables desde la línea de comandos.

Para la ingeniería, Linux ofrece tres atributos clave: **transparencia** (es posible inspeccionar procesos, recursos y políticas del sistema en tiempo real), **automatización** (la interfaz de texto, los *pipes* y los *scripts* permiten construir flujos reproducibles) y **portabilidad** (estándares, compilación con **Makefile** y código en C facilitando el control fino sobre el sistema). Estas propiedades lo convierten en el entorno natural para aprender sobre procesos, planificación, memoria y *toolchains*, y para desarrollar habilidades de diagnóstico y monitoreo que escalan desde un PC personal hasta clústeres de producción.

Un sistema operativo es, entre otras cosas, un gestor de procesos. Comprender cómo observar y analizar los procesos en ejecución es un primer paso fundamental para introducirse al mundo de Linux y los sistemas operativos modernos. En este laboratorio, exploraremos cómo capturar información en tiempo real sobre procesos activos del sistema, procesarla con herramientas de línea de comandos y encadenar scripts Bash mediante **pipes** (`|`). El objetivo no es solo aprender a usar comandos como **ps**, **awk** o **grep**, sino también entender la filosofía de Unix: “**haz una sola cosa, y hazla bien**”, componiendo soluciones más complejas a partir de programas simples.

Enunciado

En la experiencia anterior se implementaron scripts utilizando comandos Bash. Estos debieron ser capaces de:

- Generar datos del sistema sobre procesos activos mediante **ps**.
- Preprocesar los datos para limpiar y estandarizar su formato.
- Filtrar procesos según criterios configurables (uso de CPU, uso de memoria, nombre de comando).
- Transformar los datos, incluyendo la anonimización de **UIDs** de usuarios.
- Agregar métricas por comando, calculando promedios y máximos.
- Generar un reporte final con metadatos y salida en formato **CSV/TSV**.

Para esta experiencia se solicita volver a hacer uso de estos scripts, pero construyendo el pipeline de manera diferente mediante el uso y creación de procesos. Para ello, el proceso **Padre** recibe y procesa la línea de comando ingresada por **stdin**. Luego, el **Padre** crea tantos hijos como llamados de scripts contenga la línea de comando. Finalmente, por medio del uso de **pipes**, el proceso **Padre** envía la información asociada de cada script a un correspondiente proceso **Hijo**. A modo de ejemplificar lo anterior, se observa la siguiente entrada:

```
$ ./generator.sh -i 1 -t 10 | ./preprocess.sh \
```

Aquí, el **Padre** envía al **Hijo 1** los valores de las flags **-i** y **-t** para que ejecute el **generator.sh**. Luego, el **Hijo 1** le envía su salida al **Hijo 2**, para que este proceso ejecute el **preprocess.sh**. Para este caso, donde se llama a dos de los scripts, el **Hijo 2** es quién deberá mostrar la información por **stdout**.

Por otro lado, si son llamados todos los scripts, el **Hijo 6** será quién realice la salida, en este caso el **archivo.csv** con el reporte (Ver Figura 1).

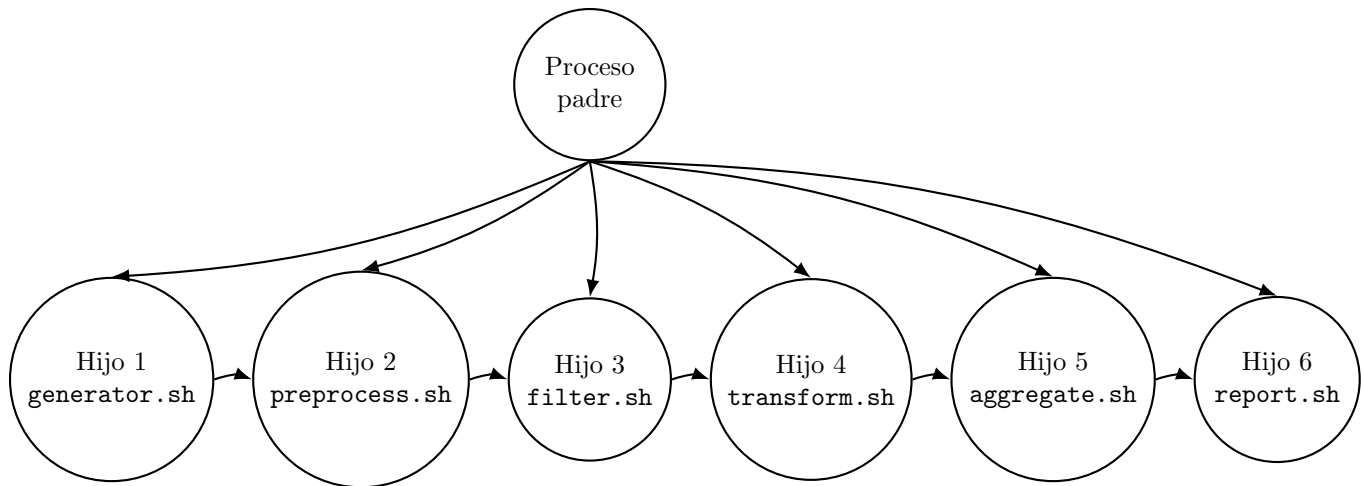


Figure 1: Diagrama donde se muestra que, dada la línea de comando donde se invocan los seis scripts, el proceso **Padre** crea un proceso **Hijo** por cada script. Luego, cada proceso **Hijo**, ejecuta el script junto con sus correspondientes flags.

Línea de comando

El flujo se compone de scripts encadenados por pipes, donde la ejecución del programa tendrá los siguientes parámetros:

```

$./lab2 generator.sh -i 1 -t 10 \
| ./preprocess.sh \
| ./filter.sh -c 10 -m 5 -r "(python|chrome)$" \
| ./transform.sh --anon-uid \
| ./aggregate.sh \
| ./report.sh -o reporte.tsv
  
```

Flags y Scripts

Cada flag corresponde a:

- **-i**: intervalo de muestreo en segundos (cada cuántos segundos generator.sh ejecuta ps).
- **-t**: duración total de la captura en segundos (tiempo que estará corriendo el generador).
- **-c**: umbral mínimo de CPU (**pcpu**); incluye procesos con $\%CPU \geq$ umbral.
- **-m**: umbral mínimo de memoria (**pmem**); incluye procesos con $\%MEM \geq$ umbral.
- **-r**: expresión regular aplicada a **comm**.

Por otro lado, los scripts realizan la siguiente tarea:

- **generator.sh**: `ps -eo pid=,uid=,comm=,pcpu=,pmem= --sort=--%cpu`, cada X segundos (**-i**), durante un tiempo total definido (**-t**).

- **preprocess.sh**: Valida el formato y los tipos de datos. Convierte opcionalmente el timestamp a ISO 8601 (`--iso8601`).
- **filter.sh**: Filtra procesos según CPU mínima (`-c`), memoria mínima (`-m`) y una expresión regular sobre el nombre de comando (`-r`).
- **transform.sh**: Anonimiza el UID con un hash (`--anon-uid`), dejando el resto de campos intactos.
- **aggregate.sh**: Agrupa por comando. Calcula: número de procesos, CPU promedio, CPU máxima, MEM promedio y MEM máxima.
- **report.sh**: Añade metadatos (`# fecha de generación, # usuario, # host`).
Escribe el resultado en un archivo CSV especificado con `-o`.

Requerimientos

Como requerimientos no funcionales, se exige lo siguiente:

- Los archivos donde estén los códigos C (los archivos `.c` y `.h`) **DEBEN** contener los nombres de los integrantes y su rut mediante el uso de comentarios.
- Debe funcionar en sistemas operativos con kernels y distribuciones Linux.
- Debe ser implementado en lenguaje de programación C.
- Se debe utilizar un archivo Makefile para compilar los distintos targets (los archivos `.c` y `.h`).
- Realizar el programa utilizando buenas prácticas, dado que este laboratorio no contiene manual de usuario ni informe, es necesario que todo esté debidamente comentado.
- Los programas se encuentren desacoplados, es decir, que se desarrollen las funciones correspondientes en otro archivo `.c` para mayor entendimiento de la ejecución.

Entregables

El laboratorio es en parejas. Si se elige una pareja, ésta no podrá ser cambiada durante el semestre. Se descontará 1 punto (de nota) por día de atraso con un máximo de tres días. A contar del cuarto, se evaluará con nota mínima. Debe subir en un archivo comprimido ZIP (una carpeta) a USACH virtual con los siguientes entregables:

- **Makefile**: Archivo para compilar los programas.
- **lab2.c**: Archivo principal del laboratorio, contiene el main para ejecutar el pipeline.
- **funciones.c**: Archivo que contiene el desarrollo de las funciones.

- **funciones.h**: Archivo que contiene las cabeceras de las funciones.
- **Código fuente de los scripts**:
 1. generator.sh
 2. preprocess.sh
 3. filter.sh
 4. transform.sh
 5. aggregate.sh
 6. report.sh

Los scripts por separado, con los nombres tal como se muestran.

- Trabajos con códigos que hayan sido copiados de un trabajo de otro grupo serán calificados con la nota mínima.
- **archivo README**: Archivo que contiene la explicación de cómo realizar la ejecución del pipeline completo. Junto con ejemplos de usos con distintos parámetros.
- **reporte.csv**: Reporte final (CSV) generado a partir de una ejecución de ejemplo.

Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje. Se deben comentar todas las funciones de la siguiente forma:

```
// Entradas: explicar qué se recibe
// Salidas: explicar qué se retorna
// Descripción: explicar qué hace
```

Además de las funciones, si su código contiene "líneas complejas", recuerde explicar lo que realizan dichas líneas, esto con el fin de lograr el entendimiento y lectura del código.

- El archivo comprimido (al igual que la carpeta) debe llamarse:

RUTESTUDIANTE1_RUTESTUDIANTE2.zip

Ejemplo 1: 19689333k_186593220.zip

- **NOTA 1**: El archivo debe ser subido a uvirtual en el apartado "Entrega Laboratorio N°2".
- **NOTA 2**: Cualquier diferencia en el formato del laboratorio que es entregado en este documento, significará un descuento de puntos.
- **NOTA 3**: SOLO UN ESTUDIANTE DEBE SUBIR EL LABORATORIO.
- **NOTA 4**: En caso de solicitar corrección del laboratorio, esta será en los computadores del DIINF, es decir, si funciona en esos computadores no hay problema.

- **NOTA 5:** Cualquier comprimido que no siga el ejemplo 1, significará un descuento de 1 punto de nota.
- **NOTA 6:** Cualquier código que NO contenga el comentario de información de los integrantes, no se revisará.

Fecha de entrega

Jueves 11 de diciembre de 2025, antes de las 23:59.