Universidad de Concepción

Tarea 1: Shell

Asignatura	Sistemas Operativos
Integrantes	Pablo Bettancourt Pinto Bastián Ceballos Zapata Angie Ramírez González Walter Zárate Solar
Profesor	Juan Felipe González

26 de septiembre 2025

1 Objetivos

Introducir a los estudiantes en el manejo de procesos concurrentes en Unix, creación, ejecución y terminación usando llamadas a sistemas fork(), exec() y wait(). Además el uso de otras llamadas a sistema como signals y comunicación entre procesos usando pipes.

2 Descripción e implementación

Para este trabajo se realizó una implementación completa de una shell con comandos personalizados, entre los cuales se incluyó "miprof", que permite ejecutar cualquier comando o programa y capturar la información respecto al tiempo de ejecución en tiempos de usuario, sistema y real más información acerca del peak de memoria máxima residente. Dicha implementación se puede encontrar dentro del siguiente repertorio de github, con sus correspondientes instrucciones de uso: https://github.com/Rhussu/Shell.

2.1 Creación de la shell

El desarrollo de la shell comenzó con la definición de un prompt interactivo que indicase al usuario que el sistema está listo para recibir comandos. Este prompt muestra el directorio de trabajo actual y reemplaza la ruta de inicio del usuario por el símbolo "~" para una presentación más compacta, además de emplear diferentes colores en cada componente para mejorar la legibilidad y apariencia. Para la entrada de comandos se utilizó la función readline(), lo que permite al usuario editar la línea de comandos de manera eficiente, manejar entradas multilínea y navegar por el historial de comandos, el cual se actualiza mediante add_history().

Para ejecutar los comandos, la shell distingue entre comandos internos (implementados o personalizados) y comandos externos. Los comandos internos se manejan directamente en la shell, para "cd" utilizamos llamadas al sistema como chdir() para cambiar de directorio y para "ls" usamos opendir() y readdir() para personalizar el listado de contenidos de un directorio; por otro lado, los comandos externos se ejecutan creando un proceso hijo mediante fork(), y posteriormente reemplazando la imagen del proceso con execvp(). Esto permite que la shell funcione de manera concurrente, ya que el proceso padre puede esperar la finalización del hijo utilizando waitpid() mientras mantiene la interacción con el usuario.

La shell cumple con otras características bases: si un comando no existe, se muestra un mensaje claro sin interrumpir la sesión y si se apreta la tecla Enter sin comandos previos, se muestra el prompt sin generar errores.

Por último, un aspecto clave de la construcción de la shell es el soporte para tuberías (pipes), en donde, para comandos encadenados, se crean múltiples procesos hijos y establece flujos de comunicación entre ellos mediante pipes. Cada proceso redirige su entrada y salida estándar

hacia el pipe correspondiente usando dup2(), asegurando que la salida de un comando sirva como entrada del siguiente.

2.2 Comando personalizado: miprof

El comando miprof identifica el modo en el que debe ser ejecutado:

1. **Modo "ejec":** Ejecuta el comando proporcionado y despliega por pantalla métricas sobre el tiempo de ejecución y el peak de memoria máxima residente luego de su ejecución.
2. **Modo "ejecsave":** Ejecuta el comando y guarda las métricas sobre el tiempo de ejecución y el peak de memoria máxima residente en el archivo proporcionado.
3. **Modo "ejecutar maxtiempo" :** Ejecuta el comando durante el tiempo proporcionado, una vez se acabe el tiempo o termine de ejecutar, despliega por pantalla métricas sobre el tiempo de ejecución del comando y el peak de memoria máxima residente luego de su ejecución.

Las mediciones de tiempo consisten en el cálculo de: tiempo real, tiempo de usuario, tiempo de sistema. El tiempo de usuario se calcula con el uso de *gettimeofday()*, a diferencia del tiempo de usuario y de sistema, que se consiguen mediante *getrusage()*, específicamente con los miembros *.ru_utime* y *.ru_stime* respectivamente, de forma similar, se consigue el valor correspondiente a la memoria máxima residente, mediante el miembro *.ru_maxrss*.

Algunos ejemplos de uso de cada comando:

- **miprof ejec sort -n numeros_1000.txt:** Imprime el sorteo en pantalla y al final entrega los tiempos/memoria.
- **miprof ejecsave resultados.txt sort -n numeros_1000.txt:** No imprime nada (más que un mensaje de confirmación de que se guardó), solo almacena los resultados de tiempo/memoria en resultados.txt o el archivo indicado
- **miprof ejecutar 5 sleep 10:** En este caso pasado los 5 segundos, detiene la ejecución de sleep aunque le falte tiempo, muestra en pantalla un aviso de que fue detenida la ejecución e imprime los tiempos y memoria hasta ese punto.
- **Extra: miprof ejecutar 5 sleep:** En caso de faltar algún argumento o que el comando a ejecutar (sleep, sort, etc.) esté incompleto, miprof no se ejecutará como tal, mostrará un aviso de que el comando está mal puesto y no entregará tiempo/memoria.

2.2.1 Sorting con comando miprof

Con la finalidad de evaluar el desempeño del comando sort utilizando el comando miprof implementado previamente, se midieron los tiempos de ejecución y el consumo de memoria para distintos tamaños de archivos de texto. Se creó una carpeta denominada miprof_sort, dentro de la cual se generaron cinco archivos con números aleatorios, con tamaños que van desde 100 hasta 1.000.000 de valores.

Cada archivo fue procesado utilizando miprof en el modo ejecsave, registrando los resultados en un archivo único de comparaciones. Los comandos ejecutados fueron de la forma *miprof ejecsave comparacion.txt sort -n archivo*, y los resultados obtenidos se resumen a continuación:

- Archivo numeros_100.txt: Tiempo real 0.0066 s, tiempo usuario 0.0014 s, tiempo sistema 0.0033 s, memoria máxima residente 2.176 KB.
- Archivo numeros_1000.txt: Tiempo real 0.0046 s, tiempo usuario 0.0050 s, tiempo sistema 0.0033 s, memoria máxima residente 2.176 KB.
- Archivo numeros_10000.txt: Tiempo real 0.0066 s, tiempo usuario 0.0072 s, tiempo sistema 0.0054 s, memoria máxima residente 2.560 KB.
- Archivo numeros_100000.txt: Tiempo real 0.0544 s, tiempo usuario 0.0511 s, tiempo sistema 0.0174 s, memoria máxima residente 7.296 KB.
- Archivo numeros_1000000.txt: Tiempo real 0.3227 s, tiempo usuario 0.9267 s, tiempo sistema 0.2873 s, memoria máxima residente 133.376 KB.

Del análisis de estos resultados se puede observar una tendencia clara: a medida que aumenta el tamaño de los archivos, el tiempo de ejecución y la memoria utilizada por el proceso crecen de manera notable. Los archivos más pequeños presentan tiempos casi despreciables y un uso de memoria mínimo, mientras que los archivos de mayor tamaño muestran incrementos significativos tanto en tiempo de CPU como en memoria, especialmente para el archivo de un millón de valores, donde se evidencia un salto importante en la memoria máxima residente.

Estos resultados reflejan la complejidad del algoritmo de ordenamiento y permiten comprender cómo se comporta sort frente a volúmenes de datos crecientes. Además, el uso de miprof facilitó la recopilación sistemática de métricas de desempeño, demostrando la utilidad de la herramienta para evaluar la eficiencia de comandos en la shell y para documentar comparaciones de forma confiable y reproducible.

3 Conclusión

La implementación de la shell permitió cumplir con los objetivos propuestos, al integrar de manera práctica conceptos fundamentales del manejo de procesos en Unix, como la creación, ejecución y terminación mediante fork(), exec() y wait(), así como la gestión de comunicación entre procesos a través de pipes. La incorporación de un prompt interactivo y comandos internos personalizados reforzó el entendimiento de las llamadas al sistema, mientras que el desarrollo del comando “miprof” ofreció un aporte adicional al permitir la evaluación del rendimiento de procesos en distintos escenarios. En este sentido, la experiencia no sólo consolidó conocimientos técnicos sobre concurrencia y llamadas al sistema, sino que también mostró la relevancia de diseñar utilidades que aporten en la medición y comparación objetiva de procesos en la shell.