Proyecto 1: Sistemas Operativos
Shell con implementación de daemon

Integrantes:
Oliver Alarcón Brito
Sofía Bravo Saavedra
Esteban Chandia Cifuentes

Introducción

Una shell es una interfaz de usuario que facilita la comunicación entre el usuario y el núcleo del sistema. Funciona como una capa interactiva que permite a los usuarios interactuar con el sistema operativo emitiendo comandos. La shell interpreta estos comandos y los traduce para que el sistema operativo los pueda ejecutar.

Un daemon es un tipo especial de programa o proceso informático que se ejecuta en segundo plano, sin intervención directa del usuario, y generalmente se inicia automáticamente cuando se inicia el sistema operativo. Los daemons son esenciales para la gestión y el funcionamiento de un sistema informático, ya que realizan diversas tareas en segundo plano, como proporcionar servicios, administrar recursos o realizar trabajos de mantenimiento.

Utilizando los conceptos anteriores, se ha desarrollado una shell simple que puede ejecutar algunos comandos comunes, y que también permite la creación de un daemon que mide y registra una línea con la información requerida del sistema en el log del sistema, todo con el objetivo de introducirse 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.

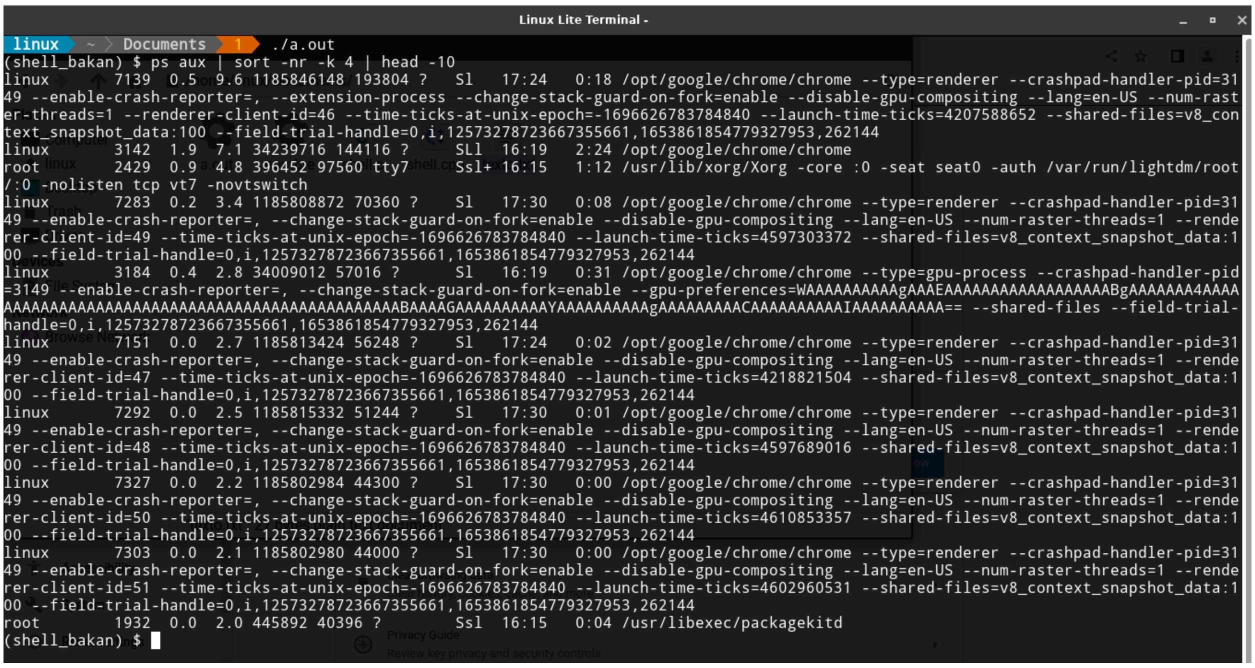
Desarrollo

Shell:

Para compilar la shell se debe ejecutar la línea: “gcc *.c -o shell_bakan” en la carpeta con todos los archivos. Después se ejecuta: “./shell_bakan”, que inicia la shell, esperando que le ingresen los comandos o pipes. Cuando la shell se encuentra en este estado, si se hace *enter* sin ingresar ningún comando, el prompt vuelve a aparecer. Los comandos que soporta esta shell son: ls, grep, echo, cat, head, tail, etc (comandos básicos de una shell según lo requerido).

Al momento de implementar la shell hay dos situaciones; la primera es cuando el input viene sin pipes. Para esto, se hacen tokens de la entrada (tokenize_input) para diferenciar entre el

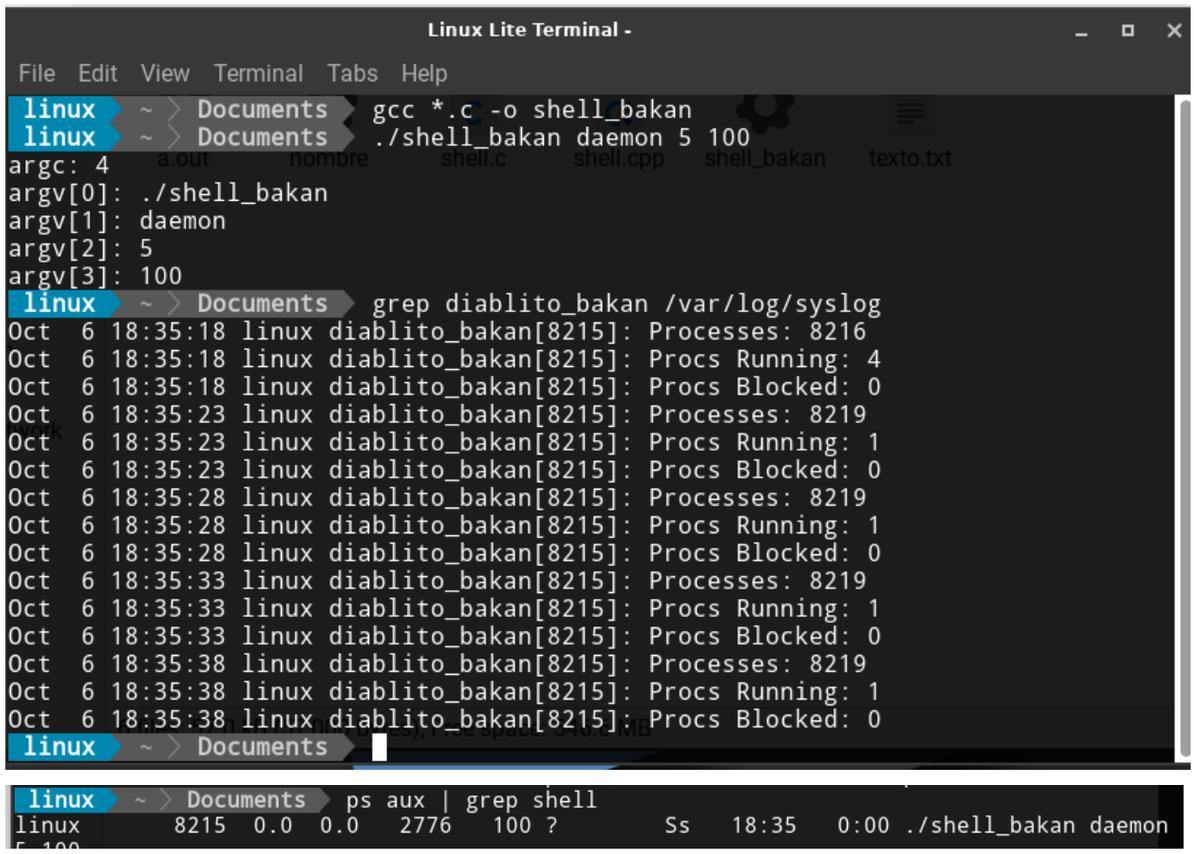
comando y sus argumentos, luego se ejecuta el comando en un proceso hijo (que se hace con fork) y con execvp (execmd). La segunda es cuando el input viene con pipes, para esto se hace parecido. Primero se separa el input con strtok para dividir los comandos separados por “|” y después se pasa a la función (execute_pipeline) que los ejecuta como pipeline. Esta función hace que, si es el primer proceso, recibe entrada estándar y manda su salida al siguiente proceso, si es un proceso intermedio, entonces recibe el input del anterior y manda su output al siguiente y ,si es el final, entonces recibe el input de otro proceso y el output es estándar..



Daemon:

Al momento de ejecutar el programa se puede ejecutar también el daemon, para esto se debe ejecutar: “./shell_bakan daemon t p”, siendo ‘t’, el intervalo de tiempo cada cuanto se ejecuta el daemon, y ‘p’, el tiempo total en el que se ejecuta el daemon . Si no se pone nada además de daemon, se tomarán por defecto los valores de t y p como 5 y 60 respectivamente. El daemon, al ser un proceso que queda ejecutándose en segundo plano, se puede verificar en la shell escribiendo el comando: “grep diablito_bakan /var/log/syslog”.

El daemon crea un proceso hijo utilizando fork y el proceso padre sale, dejando solo al proceso hijo en ejecución, de este modo queda ejecutándose en segundo plano. Luego, gracias a ‘umask’, se cambia de directorio de trabajo a la raíz del sistem y crea una nueva sesión utilizando ‘setsit()’. Luego, en syslog se abre el sistema de registro (openlog) y el daemon entra en un bucle durante ‘p’ segundos ejecutándose cada ‘t’ segundos, donde recopila y analiza la información relevante de ‘/proc/stat’ y la registra utilizando ‘syslog’. Este proceso dura hasta que haya transcurrido el tiempo total ’p’ o se puede matar antes utilizando killall shell_bakan.



Conclusión

La exitosa implementación de la shell y el daemon representa un logro significativo en este proyecto. Hemos desarrollado una shell funcional que admite todos los comandos mencionados en este informe y, al mismo tiempo, hemos logrado crear un daemon capaz de medir y registrar la información requerida de manera efectiva.

Este proyecto ha proporcionado una valiosa oportunidad para profundizar en conceptos cruciales que hemos estudiado previamente en nuestras clases. A través de la aplicación de llamadas al sistema como fork(), exec(), wait(), así como el manejo de señales y la comunicación entre procesos mediante pipes, hemos adquirido un conocimiento sólido respecto al funcionamiento de, en este caso, sistemas operativos basados en Linux, que puede ser aplicable a otros sistemas operativos.