DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

SISTEMAS OPERATIVOS

JOHN CORREDOR

TALLER 2
FORK

TOMÁS OSPINA ULLOA
DAVID SANTIAGO RODRÍGUEZ PRIETO
SEBASTIÁN SÁNCHEZ OLAYA
IVÁN CORTÉS CONSTAIN

8 DE ABRIL DE 2025
BOGOTÁ D.C.

TABLA DE CONTENIDOS

I. INTRODUCCIÓN
II. DOCUMENTACIÓN DETALLADA
III. PLAN DE PRUEBAS
IV. CONCLUSIONES
V. REFERENCIAS

I. INTRODUCCIÓN

Este taller tiene como objetivo aplicar los conceptos de procesos y comunicación entre procesos, mediante la implementación de una arquitectura que incluye el uso de fork() y pipe() para el manejo y transferencia de información entre procesos. A partir de dos archivos con arreglos de enteros, se desarrolló un programa que crea una jerarquía de procesos con el fin de calcular sumas parciales y totales de dichos arreglos, y transmitir los resultados de forma controlada y sincrónica.

II. DOCUMENTACIÓN DETALLADA

El programa desarrollado tiene como propósito realizar el procesamiento de datos provenientes de dos archivos de texto, mediante una jerarquía de procesos creada con el uso de fork() y la comunicación entre ellos a través del pipe(). Principalmente, se quiere calcular la suma de los valores contenidos en dos arreglos, leídos desde archivos distintos, y coordinar el intercambio de información entre procesos para obtener un resultado total que sea impreso por el proceso padre.

En la primera parte del código, se reciben la cantidad de elementos a leer y los nombres de los archivos correspondientes. Se utilizan funciones estándar de C como fopen() para abrir los archivos y fscanf() para leer el contenido. La información de cada archivo se guarda en arreglos dinámicos utilizando malloc(), lo que permite gestionar la memoria de forma flexible según la cantidad de datos dados por el usuario. Esta lectura se realiza por separado para los dos archivos archivo00.txt y archivo01.txt, generando así dos arreglos dinámicos independientes: arreglo00 y arreglo01.

Después, el programa establece 3 canales de comunicación mediante pipe(), el cual permite transferir los resultados parciales entre los distintos procesos. Desde allí, se inicia la creación de procesos con fork(), formando una jerarquía en la que el proceso padre engendra al primer hijo y este a su vez se comunica con el segundo hijo, es decir, el nieto del proceso principal. Esta estructura se diseña de forma que el nieto calcule la suma de los elementos de arreglo00 y le transmita el resultado al segundo hijo. El segundo hijo, por su parte, suma los valores de arreglo01, y transmite tanto su propio resultado como el del nieto al

primer hijo. Al final, el primer hijo recibe ambos resultados, los suma y envía el total, junto con los valores parciales, al proceso padre.

Cada proceso realiza un manejo cuidadoso de los descriptores de archivo, asociados a los pipes, cerrando aquellos que no utilizara para evitar errores de lectura o de escritura. Además, se hace uso del wait() para garantizar que los procesos hijos finalicen adecuadamente y evitar así la creación de procesos zombies. El proceso padre se encarga de recibir los datos en el orden correcto desde el pipe correspondiente, e imprime los tres resultados; la suma calculada por el nieto, la del segundo hijo y la suma total combinada calculada por el primer hijo.

En la parte final del código, se cierran los archivos abiertos y liberan la memoria asignada dinámicamente a los arreglos. Esta implementación no solo demuestra el uso efectivo del fork y del pipe para la comunicación entre procesos, sino también una adecuada organización de los recursos del sistema, asegurando un funcionamiento coordinado limpio en el entorno de ejecución.

III. PLAN DE PRUEBAS

Descripción del Caso	Valores de Entrada	Valor Esperado	Valor Obtenido
Tanto el fichero "*archivo00.txt*" como el fichero "*archivo01.txt*" contienen valores que están dentro de varios vectores en la ejecución del programa para reflejar un valor con la suma de los elementos en el primer fichero, otro con la suma de los elementos del segundo fichero por medio del uso de memoria dinámica, *pipes* y *fork()*. Al finalizar, se suman	● Tamaño del vector del fichero "*archivo00.txt*": 5 ● Tamaño del vector del fichero "*archivo01.txt*": 5 ● Valores del vector del fichero "*archivo00.txt*": 5 5 5 5 5 ● Valores del vector del fichero "*archivo01.txt*": 10 10 10	● Suma de los valores del fichero "*archivo00.txt*" (Nieto): 25 ● Suma de los valores del fichero "*archivo01.txt*" (2° Hijo): 50 ● Suma de ambos valores (Desde el 1° Hijo): 75	Ver Figura 1

estos dos valores para imprimirlos junto con la suma correspondiente.	10 10		

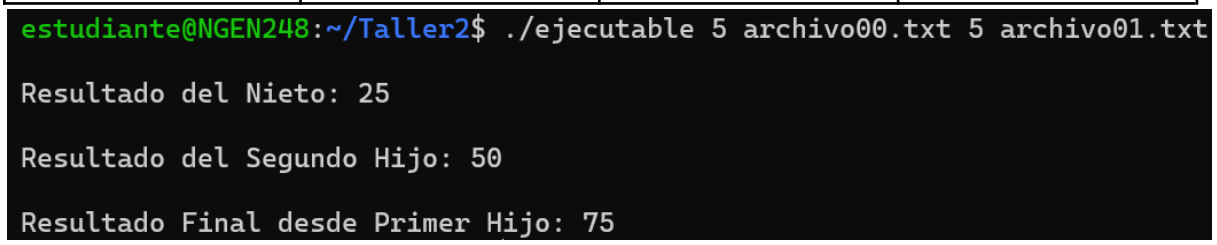

Figura 1: Resultados de las sumas después de la ejecución del programa.

IV. CONCLUSIONES

- Se comprendió del modelo de procesos en sistemas operativos mediante el código.
- Se aplicó exitosamente la comunicación entre procesos usando **pipe(),** garantizando sincronización de resultados.
- El uso de memoria dinámica permitió una gestión eficiente de los arreglos leídos desde los archivos.
- La arquitectura planteada demuestra un esquema jerárquico y ordenado que refleja un entorno de multiprocesamiento cooperativo.

V. REFERENCIAS

- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems*.
- Kerrisk, M. (2010). *The Linux Programming Interface*.
- Documentación oficial de GNU C Library: https://www.gnu.org/software/libc/
- Apuntes y clase de Sistemas Operativos - Prof. John Corredor