Pontificia Universidad Javeriana

2024


Sistemas Operativos

Taller fork

Integrantes:

Juan Ordoñez

Juan Martin Trejos

Sebastián Angulo

David Rodríguez

## 1. Resumen:

Este informe documenta el desarrollo de un programa en lenguaje C, cuyo objetivo es aplicar conceptos clave de sistemas operativos, como la creación y gestión de procesos y la comunicación entre ellos. El programa realiza la lectura de dos ficheros con arreglos de enteros, la creación de varios procesos hijos mediante el uso de la función fork(), y la implementación de mecanismos de comunicación entre procesos utilizando tuberías (pipes). El enfoque principal del taller es demostrar el manejo eficiente de la memoria dinámica, el paso de argumentos a un proceso principal y la correcta sincronización de los procesos para llevar a cabo cálculos distribuidos.

## 2. Objetivo:

El objetivo principal de este taller es aplicar los conceptos de procesos y comunicación entre procesos en sistemas operativos.

## 3. Descripción implementación:

1) **Leer arreglos de archivos:**

- El programa recibe cuatro argumentos: el tamaño y la ruta de dos archivos que contienen enteros.
- La función leer_arreglo() abre el archivo especificado, lee los enteros en un arreglo dinámico y lo retorna. Si ocurre un error en la lectura o asignación de memoria, el programa termina.

2) **Crear tuberías (pipes):**

- Tres tuberías son creadas: pipeA, pipeB, y pipeC. Estas tuberías se utilizan para pasar información entre los procesos. Las tuberías permiten que los procesos se comuniquen entre sí, ya que los datos escritos en un extremo pueden ser leídos desde el otro.

3) **Fork y procesos hijos:**

- El programa crea tres procesos con fork():
    - **Primer hijo**: Este proceso calcula la suma del primer arreglo (arregloA) usando la función calcular_suma(). Luego, escribe la suma en la tubería pipeA y finaliza.
    - **Segundo hijo**: Calcula la suma del segundo arreglo (arregloB) y escribe el resultado en la tubería pipeB. Luego, termina.

- **Tercer hijo (nieto)**: Lee las sumas de las tuberías pipeA y pipeB, calcula la suma total, y la escribe en pipeC. Después finaliza.

4) **Proceso padre:**

- El proceso principal (padre) espera a que los tres procesos hijos (incluido el nieto) finalicen.
- Luego de que todos los procesos han hecho su trabajo, el padre lee:
  - La suma de arregloA desde pipeA.
  - La suma de arregloB desde pipeB.
  - La suma total desde pipeC.
- Finalmente, imprime los resultados: la suma de ambos arreglos y la suma total.

5) **Sincronización y cierre de recursos:**

- Después de cada operación de lectura o escritura en una tubería, se cierra el lado correspondiente para liberar los recursos.
- El padre utiliza waitpid() para esperar a que los procesos hijos y el nieto terminen antes de continuar con la ejecución.

## 4. Conclusiones:

La función fork() es una herramienta esencial en los sistemas operativos Unix, como Linux, que permite la creación de nuevos procesos. Al invocar fork(), el sistema genera un proceso hijo, que es una réplica del proceso original, conocido como proceso padre. Aunque al principio ambos procesos son muy similares, hay algunas diferencias clave: Valores de retorno distintos: El proceso padre recibe el identificador del proceso hijo, mientras que el hijo recibe un valor de 0. Esta distinción es importante para que ambos puedan ejecutar tareas diferentes tras la llamada a fork(). Memoria independiente: Aunque el hijo comienza siendo una copia del padre, ambos tienen su propio espacio de memoria. Esto significa que los cambios realizados en uno no afectarán al otro, gracias a una técnica llamada "copy-on-write". Ejecución paralela: Una vez que se ha ejecutado fork(), el padre y el hijo operan de forma independiente, lo que permite que ambos se ejecuten de manera simultánea. Fundamental para la multitarea: fork() es clave para que los sistemas

operativos puedan ejecutar varias tareas a la vez, dividiendo procesos para trabajar en paralelo. En resumen, fork() permite que un proceso se divida en dos, generando un proceso hijo que puede ejecutarse junto al proceso padre. Es una técnica crucial en los sistemas operativos basados en Unix para gestionar múltiples tareas de manera eficiente y escalable.