Taller 2 – Fork ()

Juan José Ballesteros Suarez

Juan Diego Rojas Osorio

PROFESOR:

John Corredor, PhD.

Pontificia Universidad Javeriana

Facultad Ingeniería

Ingeniería de Sistemas

Sistemas Operativos

Bogotá D.C.

1. Objetivo

Los objetivos de este taller de fork() son:

- Aplicar los conceptos de creación y jerarquía de procesos haciendo uso de la función fork () en C.
- Analizar y probar la relación padre-hijo-nieto en la creación de procesos y cómo se sincronizan entre sí.
- Hacer uso de los conceptos vistos en clase y aplicarlo al ejecutar y analizar los resultados obtenidos

2. Marco Teórico

Procesos en Sistemas Operativos

Un proceso en Sistemas operativos es un programa en ejecucción, este cuenta con su propio espacio en memoria, variables y estado. Se compone de las instrucciones del programa, el estado de ejecución, datos y bloque de control de procesos.

Comunicación entre procesos

Es un conjunto de mecanismos que los sistemas operativos proveen para que los procesos puedan ser separados, compartan datos se sincronicen y se coordinen, esto se logra mediante Pipes.

Fork

Es una llamada al sistema para que cree un proceso duplicando uno existente, este denominado proceso padre. Este nuevo proceso creado se denomina proceso hijo y es una copia del proceso padre. Esta llamada al sistema fork() es útil para la creación de procesos y permite disponer de funciones importantes, como el procesamiento paralelo, multitarea y la creación de jerarquías de proceso.

Pipe

Mas conocido como Tubería, es un mecanismo de comunicación entre procesos que funciona como un canal de una sola dirección para poder intercambiar datos entre programas.

La salida de un proceso se convierte en la entrada del siguiente, esta tubería no puede cruzar su flujo, debe ser entrada o salida, no ambas

3. Desarrollo del Programa

El desarrollo de este taller lo hicimos en 3 ficheros diferentes:

- tallerFork.c: programa principal en donde se crean los procesos padre, hijo, segundo hijo y nieto
- Funciones.c: Implementa las funciones auxiliares (leer archivos, calcular sumas, y lógica de cada proceso).
- Funciones.h: Cabecera con las declaraciones de las funciones.
- Archivo00.txt y archivo01.txt: Archivos de prueba con datos específicos para poder calcular sus sumas.
- Makefile: Para la compilación del código.

Cada proceso tiene una tarea específica, en este caso el nieto se encarga de calcular la suma de los números del archivo00.txt. El segundo hijo es quien calcula la suma de los números del archivo01.txt. Para después que el primer hijo reciba ambas sumas, las combina y envía los resultados al padre. Finalmente, el padre es quien recibe las sumas y muestra el resultado final.

4. Archivos de prueba y resultados

El contenido de los archivos se prueba fueron:

Archivo00.txt: 15 30 45 60 75 90

Archivo01.txt: 20 40 60 80 100 120 140 160 180 200

Para la ejecución del codigo usamos el siguiente comando:

make

./tallerFork 5 archivo00.txt 10 archivo01.txt

Después de esto el resultado obtenido fue el siguiente:

```
estudiante@NGEN42:~/Taller02_Fork$ ./tallerFork 5 archivo00.txt 10 archivo01.txt
***** Taller Fork *****
Leyendo archivos...
Archivo 1: archivo00.txt (N1=5)
Archivo 2: archivo01.txt (N2=10)

[Segundo Hijo PID=338718] Calculando sumaB del archivo01...
[Segundo Hijo PID=338718] SumaB = 1100
[Nieto PID=338719] Calculando sumaA del archivo00...
[Nieto PID=338719] SumaA = 225
[Primer Hijo PID=338717] Calculando suma total...
[Primer Hijo PID=338717] Suma Total = 225 + 1100 = 1325

***** Resultados (Padre PID=338715) *****
SumaA (archivo00): 225
SumaB (archivo01): 1100
Suma Total: 1325
****************************************
```

Para probar esto los comprobamos sumando los datos de ambos archivos individualmente para después sumar el resultado de estos:

Archivo00.txt: 15 + 30 + 45 + 60 + 75 + 90 = 225

Archivo01.txt: 20 + 40 + 60 + 80 + 100 + 120 + 140 + 160 + 180 + 200 = 1100

Y la suma final seria: 225 + 1100 = 1325

5. Análisis

- Uso de fork() y pipe()
  El programa muestra correctamente cómo crear múltiples procesos y establecer comunicación entre ellos mediante tuberías.

  Se usan pipes unidireccionales para pasar datos entre procesos relacionados.

  Cada proceso cierra los extremos del pipe que no utiliza, evitando bloqueos o fugas.

- Sincronización con wait()
  El padre y los hijos usan wait() para garantizar que los procesos terminen en el orden correcto antes de continuar, evitando lecturas prematuras.
- Memoria y modularidad
  El código usa arreglos dinámicos (malloc) para cargar los datos desde archivo, mostrando buena práctica en el manejo de memoria, reservándola y liberando esta.

- Resultados coherentes
  Las sumas parciales y total son correctas y coinciden con los valores esperados a partir de los archivos de prueba.

6. Conclusiones

Concluyendo con el desarrollo de este taller se logró aplicar la jerarquía de procesos en C usando fork(), creando una estructura padre, hijo, nieto que muestra claramente las relaciones y herencia entre procesos.

Se implementó comunicación entre procesos mediante pipes, enviando datos sin memoria compartida, lo que demostró el concepto de comunicación entre procesos.

Con wait() se sincronizaron los procesos adecuadamente, evitando condiciones de carrera y asegurando que cada proceso espere a sus hijos antes de continuar.

Los resultados confirmaron que la suma de los archivos se calculó correctamente, validando el flujo de datos entre procesos. En general, el taller integró conceptos teóricos sobre creación, jerarquía y sincronización de procesos, reforzando la comprensión del funcionamiento de sistemas operativos con procesos concurrentes.

7. Referencias

Pipes de Linux: te explicamos el comando con ejemplos. (2022, 23 noviembre). IONOS Digital Guide. https://www.ionos.com/es-us/digitalguide/servidores/configuracion/pipes-linux

GeeksforGeeks. (2025, 23 julio). Fork System Call in Operating System. GeeksforGeeks. https://www-geeksforgeeks-org.translate.goog/operating-systems/fork-system-call-in-operating-system/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc