Taller Fork y Pipes

Andrés Loreto Quiros

Sistemas Operativos

Prof. John Corredor

Lab01

Análisis de código

El programa crea un nuevo proceso a través de la función fork(), la cual duplica el proceso actual en dos: uno es el proceso padre y el otro es el proceso hijo. Después de la llamada a fork(), ambos procesos ejecutan el mismo código, pero se diferencian gracias al valor que devuelve la función: en el padre, fork() devuelve un número positivo que representa el ID del hijo, mientras que en el hijo devuelve 0. Con base en ese valor, el programa imprime si se trata del proceso padre o del hijo, junto con su respectivo identificador de proceso (getpid()). Finalmente, ambos imprimen la línea “A partir de aquí es el proceso main o proceso principal”, demostrando que ambos ejecutan el resto del código por separado.

```
estudiante@NGEN265:~$ ./lab01
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre con ID 1539627
A partir de aquí es el proceso main o proceso principal
         #==> Proceso Hijo con ID 1539629
A partir de aquí es el proceso main o proceso principal
```

Conclusiones

Al ejecutar el programa, se observa que primero aparece el mensaje del proceso padre seguido del del proceso hijo, cada uno con un ID diferente. Esto confirma que la creación del proceso hijo fue exitosa y que ambos están corriendo de manera independiente. El orden en el que se muestran las impresiones (padre primero y luego hijo) no está garantizado: depende del planificador del sistema operativo, que decide cuál proceso se ejecuta antes. En este caso particular, el resultado se ve ordenado, pero si se ejecutara varias veces, el orden podría variar, reflejando la naturaleza concurrente de los procesos creados con fork().

Lab02

Análisis de código

El código crea un proceso hijo a partir del proceso principal usando la función fork(). Esta función duplica el proceso actual: el original se convierte en el proceso padre y la copia en el proceso hijo. Ambos ejecutan el mismo código, pero el valor que devuelve fork() permite distinguir cuál es cuál: en el hijo devuelve 0 y en el padre un número positivo (el ID del hijo). Luego, cada uno imprime sus mensajes y un ciclo con los números del 0 al 4. Como ambos procesos corren en paralelo, el sistema operativo decide cuál se ejecuta primero y cuándo alternar entre ellos, lo que puede hacer que las líneas de texto se entremezclen en el resultado. En este caso solo se usa fork(), no se utilizan pipes; si se quisieran comunicar el padre y el hijo, sería necesario implementar un pipe para enviar información entre ambos procesos.

```
estudiante@NGEN265:~$ gcc lab02.c -o lab02
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
```

```
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==
                  == 3 ==
                  == 4 ==
FIN ....
         #==> Proceso Hijo recién Creado
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==
                  == 3 ==
                  == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
Imprimiendo ....
         #==> Proceso Hijo recién Creado
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==
                  == 0 ==
                  == 3 ==
                  == 1 ==
                  == 4 ==
                  == 2 ==
FIN ....
                  == 3 ==
                  == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
Imprimiendo ....
         #==> Proceso Hijo recién Creado
                  == 0 ==
Imprimiendo ....
                  == 1 ==
```

```
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
         #==> Proceso Hijo recién Creado
Imprimiendo ....
Imprimiendo ....
                  == 0 ==
                  == 0 ==
                  == 1 ==
                  == 1 ==
                  == 2 ==
                  == 2 ==
                  == 3 ==
                  == 3 ==
                  == 4 ==
                  == 4 ==
FIN ....
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==
                  == 3 ==
                  == 4 ==
FIN ....
         #==> Proceso Hijo recién Creado
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==
                  == 3 ==
                  == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


         #==> Proceso Padre
Imprimiendo ....
                  == 0 ==
                  == 1 ==
                  == 2 ==

estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
        #==> Proceso Hijo recién Creado
                == 0 ==
Imprimiendo ....
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
```

```
FIN ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 0 ==
                == 3 ==
                == 1 ==
                == 4 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
FIN ....
estudiante@NGEN265:~$ ./lab02
#=> Inicio del proceso main o proceso principal <=#


        #==> Proceso Padre
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
        #==> Proceso Hijo recién Creado
Imprimiendo ....
                == 0 ==
                == 1 ==
                == 2 ==
                == 3 ==
                == 4 ==
FIN ....
estudiante@NGEN265:~$
```

Conclusiones

Al comienzo se puede ver que la ejecución da resultados constantes y, al ejecutarlo varias veces seguidas, empieza a haber cierto “caos” en el orden de impresión porque el padre y el hijo corren en paralelo sin sincronización.

Esto se debe a las decisiones del planificador del sistema operativo y a que ambos procesos escriben concurrentemente en la misma salida estándar; por eso las líneas se intercalan de forma no determinista.

Lab03

Análisis de código

El programa crea un pipe (fd[0] lectura, fd[1] escritura) y luego llama a fork() para duplicar el proceso. El padre cierra su extremo de lectura, escribe en el pipe el mensaje "Hola HIJO desde el PADRE!!" y cierra el extremo de escritura. El hijo cierra su extremo de escritura, lee del pipe hacia buffer, cierra el extremo de lectura y muestra en pantalla el mensaje recibido. Finalmente, el padre hace wait(NULL) para esperar a que el hijo termine y luego imprime FIN.....

```
estudiante@NGEN265:~$ ./lab03
 #=> Inicio del proceso main o proceso principal <=#

        ##==> Proceso HIJO: recibe mensaje del PADRE => Hola HIJO desde el PADRE!!
FIN.....
FIN.....
```

Conclusiones

Gracias a wait(NULL) y al cierre correcto de extremos del pipe, el orden de salida se vuelve lógico y estable: primero verás el mensaje del hijo mostrando lo que recibió del padre (##==> Proceso HIJO: recibe mensaje del PADRE => ...) y, sólo después, el padre imprime su FIN..... El cierre de fd[1] en el padre es clave para que la lectura del hijo termine (reciba EOF) y no se quede bloqueado; del mismo modo, cerrar fd[0] en el padre y fd[1] en el hijo evita fugas y lecturas/escrituras accidentales. En resumen, el pipe transporta el mensaje unidireccionalmente del padre al hijo y wait() sincroniza el final, produciendo una salida predecible sin intercalado caótico.