SOPER: Memoria práctica 1

Darío Adrián Hernández Barroso
Ángel Manuel Martín Canto

2016-02-25

Descriptores de fichero

La parte de la practica que trataba exclusivamente sobre descriptores de fichero y su uso no contenía ningún ejercicio entregable. Sin embargo, a modo de aprendizaje y asentamiento de conocimientos realizamos un clon sencillo de la utilidad cat de UNIX. Dicho programa se limita a concatenar los archivos que recibe como argumentos(o la entrada estándar en caso de no recibir argumentos) y mostrarlos por salida. Con el uso de redirecciones sirve como editor primitivo para leer y escribir archivos. Adjuntamos el código fuente como cat.c

Procesos y fork

Ejercicio 4 original

En el ejercicio 4 hemos añadido salidas por pantalla por mostrar los PIDs de todos los procesos, los PPIDs de los hijos y el momento en el que cada proceso muere. También hemos añadido sleeps en los hijos y antes de salir para poder ver el arbol de procesos desenvolverse. Con solo estos cambios la salida de el programa es:

```
PID = 1094
PADRE 1094
PADRE 1094
HIJO 1207 CON PADRE 1094
PADRE 1094
HIJO 1206 CON PADRE 1094
HIJO 1208 CON PADRE 1094
MUERE 1094
PADRE 1206
HIJO 1361 CON PADRE 1206
HIJO 1362 CON PADRE 1207
PADRE 1206
HIJO 1363 CON PADRE 1206
MUERE 1208
MUERE 1207
MUERE 1206
PADRE 1361
HIJO 1390 CON PADRE 1361
MUERE 1362
MUERE 1363
MUERE 1361
MUERE 1390
```

Un “pswait -p 1094” durante la ejecución del programa llega a mostrar lo siguiente:

```
ejercicio4(1094)-+-ejercicio4(1206)
                 +-ejercicio4(1207)
                 +-ejercicio4(1208)
```

Y sucesivas llamadas no muestran nada. Como se ve en la salida del programa el proceso 1094 es el primero en morir, dejando al resto del árbol huérfano e impidiendo que pstree vea el árbol a partir del PID 1094.

Usando “pswait | grep ejercicio4” múltiples veces se ven fragmentos huérfanos apareciendo y desapareciendo.

Con un wait()

Tras añadir un wait la salida del programa es la siguiente:

```
PID = 2481
PADRE 2481
HIJO 2604 CON PADRE 2481
PADRE 2481
PADRE 2481
HIJO 2605 CON PADRE 2481
HIJO 2606 CON PADRE 2481
PADRE 2604
PADRE 2604
HIJO 2774 CON PADRE 2604
PADRE 2605
HIJO 2773 CON PADRE 2604
HIJO 2775 CON PADRE 2605
MUERE 2606
MUERE 2481
HIJO 2924 CON PADRE 2773
MUERE 2774
MUERE 2775
MUERE 2604
MUERE 2605
MUERE 2924
MUERE 2773
```

Sigue habiendo procesos huerfanos, por ejemplo 31231, que muere de los últimos pero cuyo padre, 31157 muere antes.

La ejecución de "pstree -p 2481" muestra diversos arboles incompletos antes de no poder mostrar más por la muerte del proceso raíz:

```
ejercicio4(2481)-+-ejercicio4(2604)-+-ejercicio4(2773)
                 |                  +-ejercicio4(2774)
                 +-ejercicio4(2605)---ejercicio4(2775)
```

La razón es que wait() espera a un solo proceso hijo pero cada proceso genera hasta tres hijos, por lo que dos se pueden quedar huérfanos.

Con el número necesario de wait()'s

Tras arreglar esto, ejecutando wait() tantas veces como hijos se crean(notese que cada proceso hace de 0 a 3 hijos) la salida del programa es la siguiente:

```
PID = 3097
PADRE 3097
PADRE 3097
PADRE 3097
HIJO 3171 CON PADRE 3097
HIJO 3170 CON PADRE 3097
HIJO 3172 CON PADRE 3097
PADRE 3171
HIJO 3304 CON PADRE 3171
PADRE 3170
HIJO 3305 CON PADRE 3170
PADRE 3170
HIJO 3306 CON PADRE 3170
MUERE 3172
```

```
PADRE 3305
HIJO 3441 CON PADRE 3305
MUERE 3304
MUERE 3306
MUERE 3171
MUERE 3441
MUERE 3305
MUERE 3170
MUERE 3097
```

Revisando la salida vemos todas las muerte ordenadas y contamos que son 8, esto es $NUM\_PROCS^2 - 1 = 3^2 - 1 = 8$, justo lo que esperamos de un fork() en un bucle de que se ejecuta NUM_PROCS veces en la raiz.

En el código fuente que entregamos se puede cambiar el numero de WAITs comentando o descomentando macros. Por defecto se esperan todas las veces necesarias.

Ejecución de programas - Llamadas exec()

El único ejercicio entregable de está seccion es el número 8. Simplemente usamos la función execvp(), que permite pasar argumentos para transmutar el proceso en "du –apparent-size -BK <nombre_del_ejecutable>". A partir de ese punto du imprime el tamaño en KB del ejecutable. Tras execvp() solo hay código para generar un error dado que nunca se debería llegar a ejecutar esa sección si exec funciona.

Comunicación entre procesos con tuberías

El ejercicio 9 requiere comunicar procesos de una jerarquía mediante tuberías. La comunicación es en los dos sentidos pero el orden de turnos para hablar está bien definido. Valoramos la solución de generar dos tuberías por proceso hijo, una para hablar de hijo a padre y otra para hablar de padre a hijo. En vez de eso, reutilizamos una tubería en los dos sentidos, primero habla el padre, después el padre espera a que el proceso hijo termine(este es otra requisito del ejercicio) y por último lee de la tubería con la garantía de que el hijo ya ha escrito. Con esta sincronización provista por wait() el programa funciona con un tubería por hijo. La salida del programa es:

```
Padre@9157 nace
Hijo@9158 nace
Hijo@9159 nace
Hijo@9158:Datos enviados a través de la tubería por el proceso PID=9157
Hijo@9159:Datos enviados a través de la tubería por el proceso PID=9157
Nieto@9160 nace
Nieto@9160:Datos enviados a través de la tubería por el proceso PID=9158
Nieto@9161 nace
Nieto@9160 muere
Nieto@9162 nace
Nieto@9161:Datos enviados a través de la tubería por el proceso PID=9158
Nieto@9162:Datos enviados a través de la tubería por el proceso PID=9159
Nieto@9162 muere
Nieto@9163 nace
Nieto@9161 muere
Nieto@9163:Datos enviados a través de la tubería por el proceso PID=9159
Nieto@9163 muere
Hijo@9158:Nieto@9160 terminado con 0
Hijo@9159:Nieto@9162 terminado con 0
Hijo@9159:Datos enviados a través de la tubería por el proceso PID=9162
Hijo@9158:Datos enviados a través de la tubería por el proceso PID=9160
```

```
Hijo@9158:Nieto@9161 terminado con 0
Hijo@9159:Nieto@9163 terminado con 0
Hijo@9158:Datos enviados a través de la tubería por el proceso PID=9161
Hijo@9159:Datos enviados a través de la tubería por el proceso PID=9163
Hijo@9158 muere
Hijo@9159 muere
Padre@9157:Hijo@9158 terminado con 0
Padre@9157:Datos enviados a través de la tubería por el proceso PID=9158
Padre@9157:Hijo@9159 terminado con 0
Padre@9157:Datos enviados a través de la tubería por el proceso PID=9159
Padre@9157 muere
```

El envío de mensajes y la creación y muerte de procesos se realiza en el orden correcto.