Memoria de la práctica 1
Sistemas operativos 2018-2019

Alejandro Pascual y Víctor Yrazusta

27 de febrero de 2019

Respuestas a las preguntas cortas

Ejercicio 3

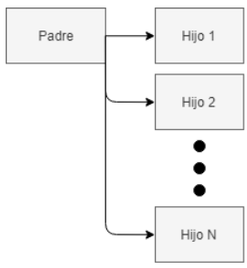
a) Analiza el texto que imprime el programa. ¿Se puede saber a priori en qué orden se imprimirá el texto ? ¿Por qué?

No se puede saber cual se ejecuta antes, ya que el padre no espera a la ejecución del hijo.

b) Cambia el código para que el proceso hijo imprima su PID y el de su padre en vez de la variable i.

Lo hemos hecho usando las funciones *getpid()* y *getppid()*.

c) Analiza el árbol de procesos que genera el código de arriba. Muéstralo en la memoria como un diagrama de árbol (como el que aparece en el ejercicio 4) explicando porqué es así.



El padre va creando procesos hijos y no espera a que estos terminen su ejecución para terminar él la suya, de manera que deja hijos huérfanos.

Ejercicio 4

a) El código del ejercicio 3 deja procesos huérfanos, ¿Por qué?.

El padre puede dejar 2 hijos huérfanos al solo tener un *wait()* e iniciar 3 procesos hijo.

b) Introduce el mínimo número de cambios en el código del ejercicio 3 para que no deje procesos huérfanos.

Los cambios se limitan a introducir un *wait()* dentro del bucle en el que el padre va creando procesos.

Ejercicio 5

a) En el programa anterior se reserva memoria en el proceso padre y la inicializa en el proceso hijo usando el método strcpy(que copia un stringa una posición de memoria), una vez el proceso hijo termina el padre lo imprime por pantalla. ¿Qué ocurre cuando ejecutamos el código? ¿Es este programa correcto? ¿Por qué? justifica tu respuesta.

El padre no imprime el valor al que el hijo ha inicializado la variable. Suponemos que esto se debe a que, cuando se ejecuta *fork()*, se trata de la misma forma a la memoria dinámica y a la estática, generando una nueva copia para el proceso hijo.

b) El programa anterior contiene un memory leak ya que el array sentence nunca se libera. Corrige el código para eliminar este memory leak. ¿Dónde hay que liberar la memoria en el padre, en el hijo o en ambos?. Justifica tu respuesta.

La respuesta va ligada al punto anterior, al hacer una nueva copia de la memoria dinámica, es necesario liberarla dos veces, una en cada proceso.

Detalles de implementación

Ejercicio 3

Nos limitamos a implementar los cambios funcionales especificados modificando lo menos posible el código.

Ejercicio 4

Nuestro programa tiene un proceso padre que crea una cantidad determinada de procesos hijos. Todos los procesos esperan a todos sus hijos e imprimen si son padres o hijos.

Ejercicio 7

En este ejercicio hemos tenido que añadir dos comandos shell al código ya dado, estos se ejecutan en procesos independientes. En el caso de "ls" utilizamos la función execlp(), que indica que el programa estará en path y que los datos se pasarán como lista de argumentos. Para "cat" utilizamos execvp(), que también indica que el programa se situa en path, pero en este caso los argumentos son pasados en un array.

Ejercicio 9

Primero creamos los procesos y los pipes, y nos cercioramos de que ambos se hayan creado correctamente. Después cada proceso ejecuta su funcionalidad, valiéndose de pipes para pasarse la información entre ellos.

Ejercicio 12

Primero creamos los hilos e inicializamos la matriz de argumentos, a partir de la cual se calcularán las potencias de 2. Después sincronizamos el proceso principal con el resto de hilos e imprimimos los resultados que han calculado.

Para el cálculo de las potencias de 2 hemos usado el operador $\ll$ (bitwise left shift), que desplaza el número dado una cantidad de posiciones hacia la izquierda en binario. Como cada desplazamiento equivale a multiplicar el número por 2, simplemente lo desplazamos N posiciones para calcular 2^N. Hemos usado este método por que resulta mucho más eficiente que multiplicar el número repetidamente por 2, aunque en función del compilador este podría realizar también esta optimización.