Especificación de la utilidad *lanzador_temporizado*

NOMBRE

lanzador_temporizado

UTILIZACIÓN

./lanzador_temporizado tiempo comando1 … comando5

DESCRIPCIÓN

Lanzador_temporizado recibe un número de segundos y entre 1 y 5 comandos o programas sin argumentos como podría ser *date, ls, ps o who*.

Estos programas o comandos se lanzarán de manera concurrente. Cuando un programa o comando termine, la utilidad imprimirá una línea en la salida estándar indicándolo. Transcurrido el número de segundos pasado como primer argumento, la utilidad forzará la terminación de aquellos comandos/programas que no hubieran terminado aún, indicándolo igualmente en la salida estándar.

En cuanto todos los comandos/programas hayan terminado, la utilidad escribirá en la salida estándar la duración total en segundos del experimento y finalizará su ejecución.

VALORES DE RETORNO

En las siguientes situaciones no mostrará el resultado esperado, ya sea el tiempo de todos los comandos terminados o mensaje del tiempo acabado. Se informará con un mensaje de error, informando además por la salida estándar de errores un mensaje. indicando el tipo de error:

1. Número de argumentos erróneos. El cual debe ser entre dos y 7 argumentos.

2. El segundo parámetro pasado debe de ser un número.

3. Error al llamar a fork() para crear un proceso hijo.

4. Programa o comando no existente a la hora de pasarlo al exec.

5. Otro tipo de errores.

En caso contrario (si no hay errores) devolverá 0, si no -1.

VER TAMBIÉN

fork(2), exec(3), time(1), wait(3), kill(1)

Descripción del "algoritmo" lanzador_temporizado

Primeramente en nuestro código realizamos un control de errores:

1. Controlamos que el número de parámetros esté dentro de lo establecido.
2. Controlamos además que el parámetro debe de ser un número.

Tras eso empezamos a contar el tiempo para después mostrarlo en caso de que los procesos terminen antes que el reloj.

Lanzamos un proceso hijo clock, llamado *clockid* con el que llamamos a un programa externo *./clock* pasándole el primer parámetro por medio de *execlp*. Realizamos un control por si hay algún problema al crear el subproceso.

Posteriormente realizamos otro proceso hijo, *childid*, el cual será el encargado de escribir por pantalla cada vez que su subproceso termine de realizar el comando o programa pasado.

Como he mencionado vamos a realizar tantos subprocesos como comandos o programas pasados por parámetro, entonces realizamos un bucle para poder escribirlos. Se van a ejecutar secuencialmente y cada uno llamará a su programa/comando por medio de *execlp*.

Childid realizará también un bucle para poder esperar a que cada uno de los procesos terminen.

Si el termina antes el reloj, se mata al proceso *childid* y se imprime un mensaje de tiempo excedido.

En caso contrario, se mataría el proceso reloj y se imprimiría el tiempo total transcurrido.