Joan Gracia Taribó - 219176
Josep Tuñí Domínguez - 216737

3. Fork & Exec

1.

a) ./main

En este caso se ejecuta main sin ningún otro comando, por tanto se crea el proceso del padre y a continuación el proceso del hijo, en este caso, el proceso hijo no hará nada ya que el número de argumentos es uno (./main), por lo que el primer if de nuestro proceso hijo no se va a cumplir y lo siguiente que se hará será terminar el proceso hijo con código 1. Después de que el hijo acabe, se recuperará el código de salida de nuestro proceso hijo para saber desde el padre si nuestro hijo ha ejecutado algún comando o no, en este caso el código del hijo es 1, por tanto, no se ha ejecutado ningún comando en el hijo. En total se crean 2 procesos.

b) ./main ls

En este caso se ejecuta main, que creará el proceso padre y luego al hijo, el programa tiene más de un argumento, por lo que nuestro proceso hijo ejecutará el comando ls, el comando ls nos listará los archivos y los directorios del directorio actual, una vez el proceso hijo haya ejecutado el comando ls terminará con el código de salida 2 para indicar que se ha ejecutado algo. Este código será recuperado por el proceso padre, que al ver que el hijo ha terminado, terminará también. En total se crean 2 procesos.

c) ./main ls -la

Nuestro programa creará el proceso padre y luego al hijo, que ejecutará ls con la flag -la, esto nos listará archivos y carpetas de nuestro directorio y para cada uno nos dirá los permisos, el propietario, la fecha de creación y el tamaño. Se crean 2 procesos.

d) ./main ps

Nos va indicar que procesos se están ejecutando (Ligados a la terminal) en este momento, en este caso se estará ejecutando el proceso de la terminal, el main y el comando ps que es ejecutado por el proceso hijo. Se crean 2 procesos.

e) ./main ps aux

En este caso veremos todos los procesos que se están ejecutando en este momento (Ligados y no ligados a la terminal) y el usuario que controla ese proceso. Se crean 2 procesos.

f) ./main < cmds.txt

g) ./main < ls

Aquí tenemos un error, después de < debe venir un archivo .txt que se va a pasar a la standard input, en este caso le estamos dando un comando, que realmente es una ruta a un ejecutable. No se crea ningún proceso.

h) ./main &

En este caso se ejecuta main, se crea el proceso hijo y como hay el carácter &, esperará a tener otra instrucción. En este caso se crean dos procesos y si después del & se llama a otra instrucción, se creará otro proceso para esa instrucción. Se crean dos procesos o más.

i) ./main lss

El comando lss no existe en UNIX, por lo que se creará el proceso padre, luego el hijo y este último proceso intentará ejecutar la instrucción ‘lss’ que como no existe, no hará nada y devolverá el código de salida 2, de esta forma el padre sabrá que el hijo no ha encontrado el comando. Se crean 2 procesos.

j) ./main ./main

En este caso se crea el padre del primer main, que creará un proceso hijo que ejecutará el comando ./main, este segundo main creará un nuevo proceso que creará otro proceso hijo, que como no tendrá ningún comando, no hará nada. En total, se crean 4 procesos.

k) ./main ./main ls

Este caso es muy similar al anterior, pero en este caso, el último proceso si que tendrá un comando que ejecutar, así que el último hijo ejecutará el comando ls que nos listará los archivos y directorios del directorio actual. Se crean 4 procesos en total.

l) ./main ./main ls -la

En este caso va a ocurrir lo mismo que en k) pero se van a listar los archivos del directorio y para cada uno nos dirá los permisos, el propietario, la fecha de creación y el tamaño. Se crearán 4 procesos.

Joan Gracia Taribó - 219176
Josep Tuñí Domínguez - 216737

2.
Si nuestro proceso hijo llama al execvp con un comando correcto, automáticamente llamará al _exit(0), eso significa que nuestro comando se ha ejecutado correctamente. En caso de que el comando no se encuentre, execvp no podrá ejecutar nada y no llamará a _exit, por lo que tendremos que hacer esa llamada nosotros mismo para poder mandar el código de error, indicando que no se ha encontrado ese comando. Haremos _exit(2).

3.
Si no se cumple la condición argc > 1, significa que solo hemos ejecutado ./main, por lo que no hay ningún otro comando que ejecutar, por lo que no se llamará a la función execvp y no tendremos ninguna información sobre el proceso hijo, añadiendo este _exit(1), nos aseguramos que si se llega a ese punto del código es que no se ha entado en el if y que por tanto, no se ha cerrado el proceso, así que cerramos nosotros el proceso, con el código 1 para indicar que no se ha ejecutado ningún comando.