**Trabajo Práctic0 04**

**Programas concurrentes (padre-hijos-nietos).**

**OBJETIVO.**

Crear varios procesos hijos a partir de un proceso padre que deberá esperar su respectiva terminación, tal y como lo hace el Sistema Operativo en su calidad de Programa Concurrente.

**INTRODUCCIÓN**

En el sistema operativo de Unix se puede utilizar la llamada a fork(), lo cual hace que el proceso que se ejecuta se divida en dos procesos. Al proceso que ejecuta fork se le conoce como padre y al nuevo proceso se le llama proceso hijo. Algunas de las llamadas al sistema que nos permiten manipular los procesos son: exec(), para ejecutar un nuevo programa, wait(), para que un proceso padre sincronice su ejecución con la llamada exit de un proceso hijo, y exit(), que termina la ejecución de un proceso. En este trabajo utilizaremos el visor VMware para manipular programas concurrentes con scripts de C++ y la terminal de Unix.

**DESARROLLO**

En el archivo ***CHeaders.xlsx*** puede usted encontrar algunos de los Headers de C++ para Linux/Unix. Para el caso de ***fork()*** y la familia ***exec…(…)*** lea el archivo ***Fork&Execfam.pdf***, en el fólder de la clase anterior.

Dada la situación de ejecución de programas en su directorio de trabajo es fundamental que definan él “.” (punto) al inicio del *PATH*.

1. Creación de procesos hijos con ***fork()***, y ***exec…(…)***.

Con programas al estilo de la clase pasada, un proceso padre deberá crear tres procesos hijos y esperar a que dichos hijos terminen.

Dentro de este fólder se encuentra el directorio *edoini* donde se encuentran los archivos *padrea.cc*, *hijofina1.cc*, *hijofina2.cc*, y *hijofina3.cc*. Por el momento *padrea* sólo lanza la ejecución del proceso hijo *hijofina1*. Los tres programas hijo tienen programado un ciclo que se repite una cantidad de veces random, esto con el objetivo que al ser ejecutados los procesos hijos terminen en distinto orden, cada vez que se ejecuten todos. También traen programado el mandar ejecutar el comando *ps*.

Entonces el programa *padrea.cc* deberá crear, via sus correspondientes ***fork()***, tres procesos clon hijo, donde cada uno. vía su respectivo ***exec…(…)*** lanzará la substitución por los programas hijos.

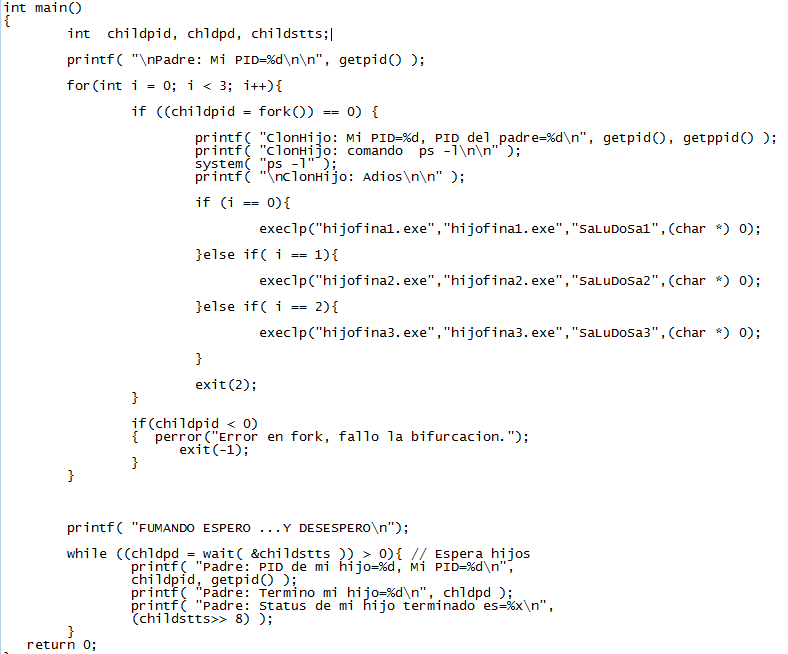
Siempre es conveniente que se compilen todos los programas después de modificar alguno de ellos o todos, para esto se les provee con el archivo *pahib.src* que se debe aplicar como *source pahib.src*, logrando así los ejecutables padre e hijos finales.

Antes que nada examine los códigos de los programas.

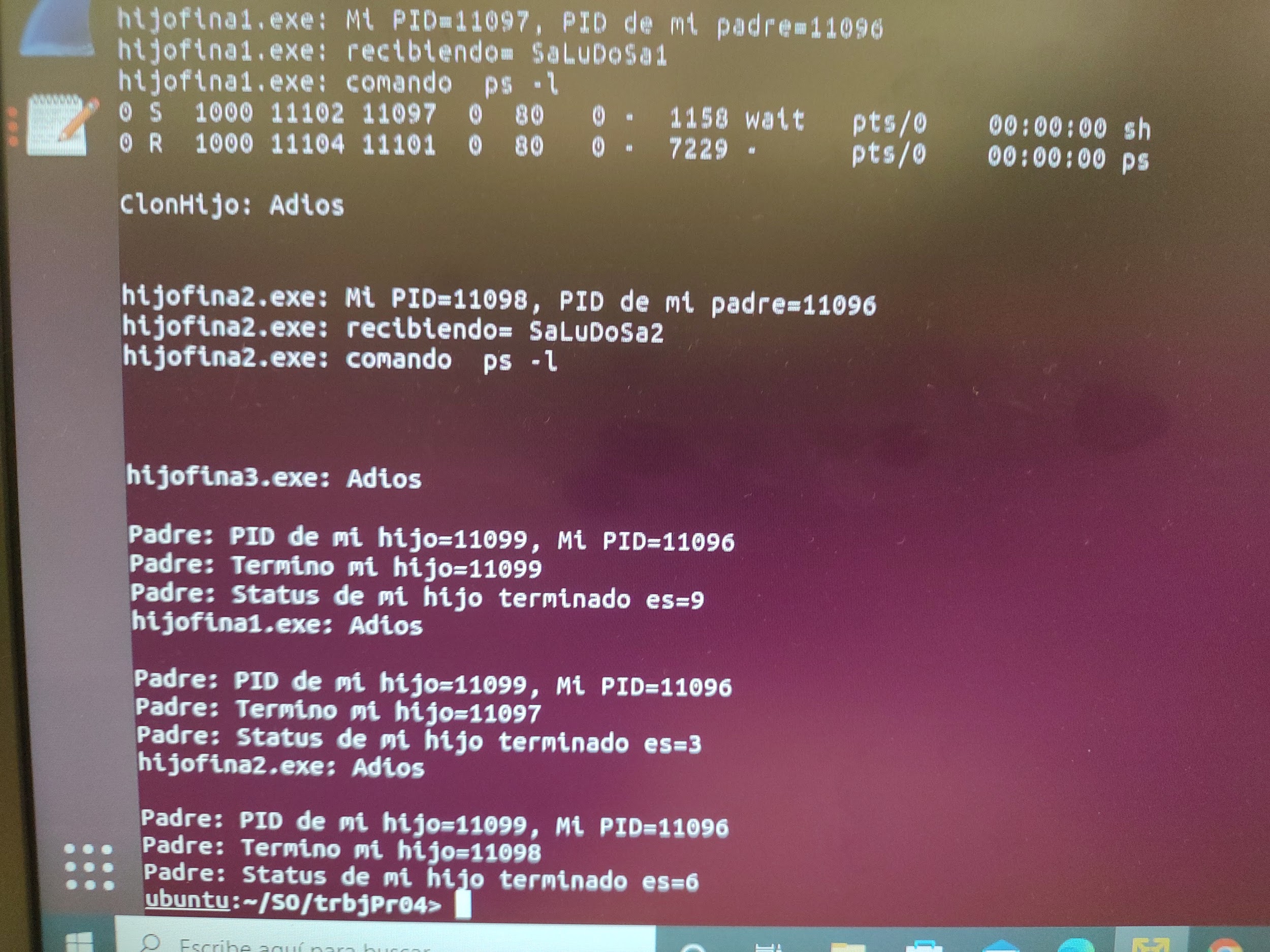
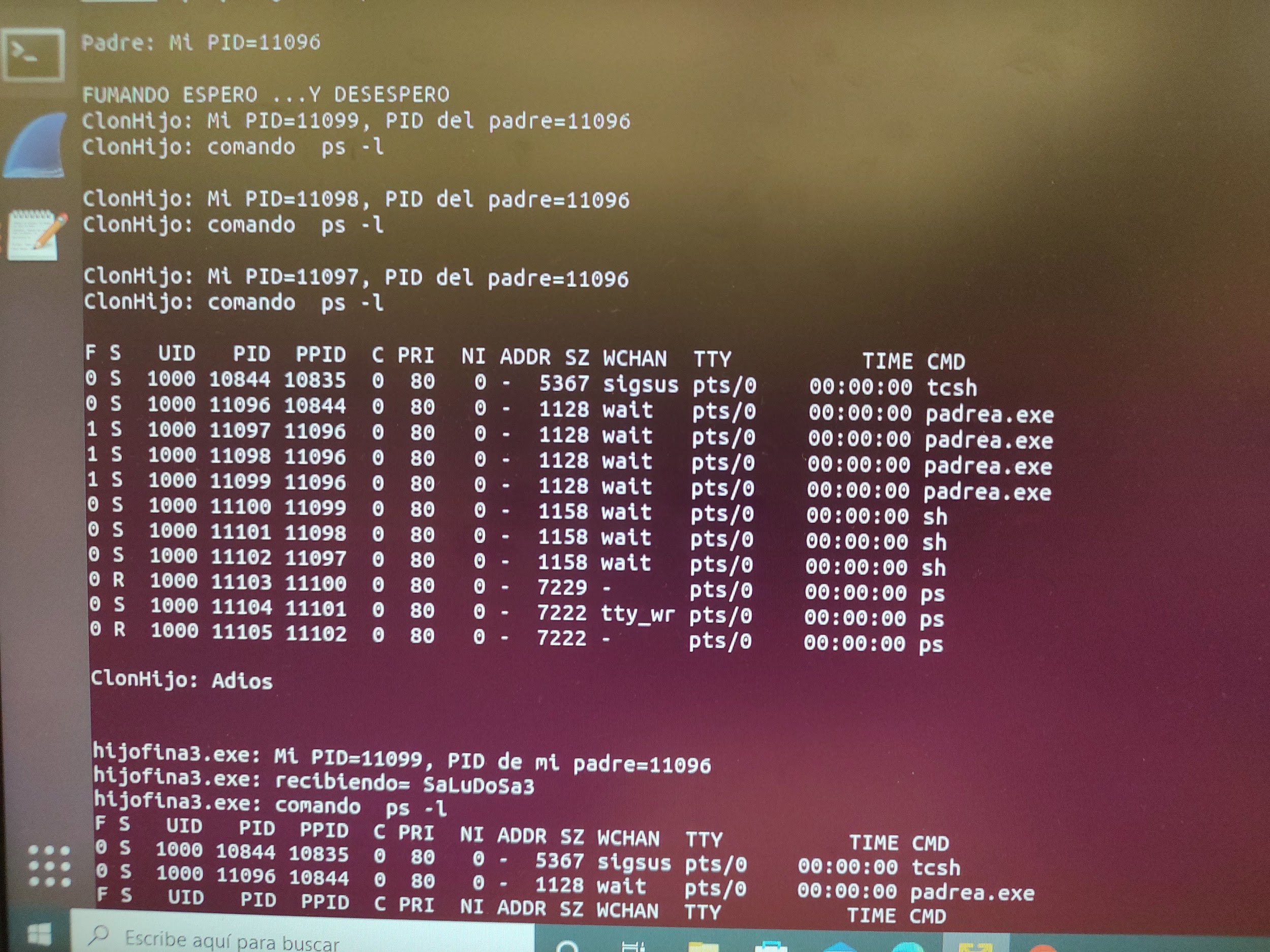
Siempre se debe imprimir, al final, el valor estatus que envía cada proceso hijo al terminar.

Despliegue un “pantallazo” o “screenshot” de la corrida, ya con los tres hijos.

**Se modificaron las siguientes líneas del programa padrea.cc para poder mostrar sólo tres procesos clon hijo.**

****

**A continuación se muestran las capturas de pantalla de los tres hijos creados.**



Dentro del programa ***padrea.cc*** tiene los siguientes argumentos en ***excelp***

execlp("hijofina1.exe","hijofina1.exe","texto-del-programador",(char \*) 0);

execlp("hijofina2.exe","hijofina1.exe","texto-del-programador",(char \*) 0);

execlp("hijofina3.exe","hijofina1.exe","texto-del-programador",(char \*) 0);

donde el string “texto-del-programador” es pasado al arrancar cada uno de los hijos "hijofina1.exe", "hijofina2.exe" e "hijofina3.exe". Este string “texto-del-programador” debe ser distinto en cada hijo.

Cada hijo deberá generar un respectivo archivo de texto (hijofina1.txt, hijofina2.txt e hijofina3.txt) donde deberá guardar tanto el string pasado a cada hijo, como el despliegue del comando “ps –l” en cada hijo.

**SIGUIENTE PÁGINA >>**

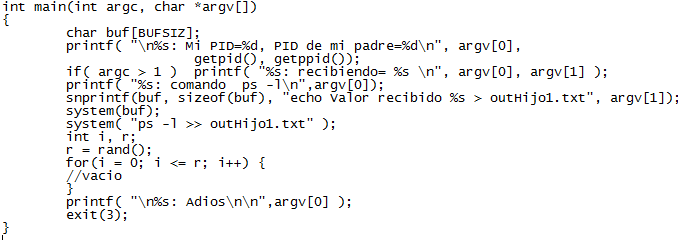
De los argumentos del *excelp* antes mostrado, explique para este caso, cual es el significado y la utilidad, de cada uno de los 4 argumentos.

**El comando execlp utilizado contiene 4 argumentos: el primer argumento sirve para ejecutar el programa o filename deseado dentro del path; el segundo como el tercer argumento sirven para caracterizar una lista de strings sobre el programa en ejecución, y estos 2 argumentos en particular se pasan a los programas “hijofina”; finalmente, el último argumento sirve para indicar que las lista de argumentos se termina.**

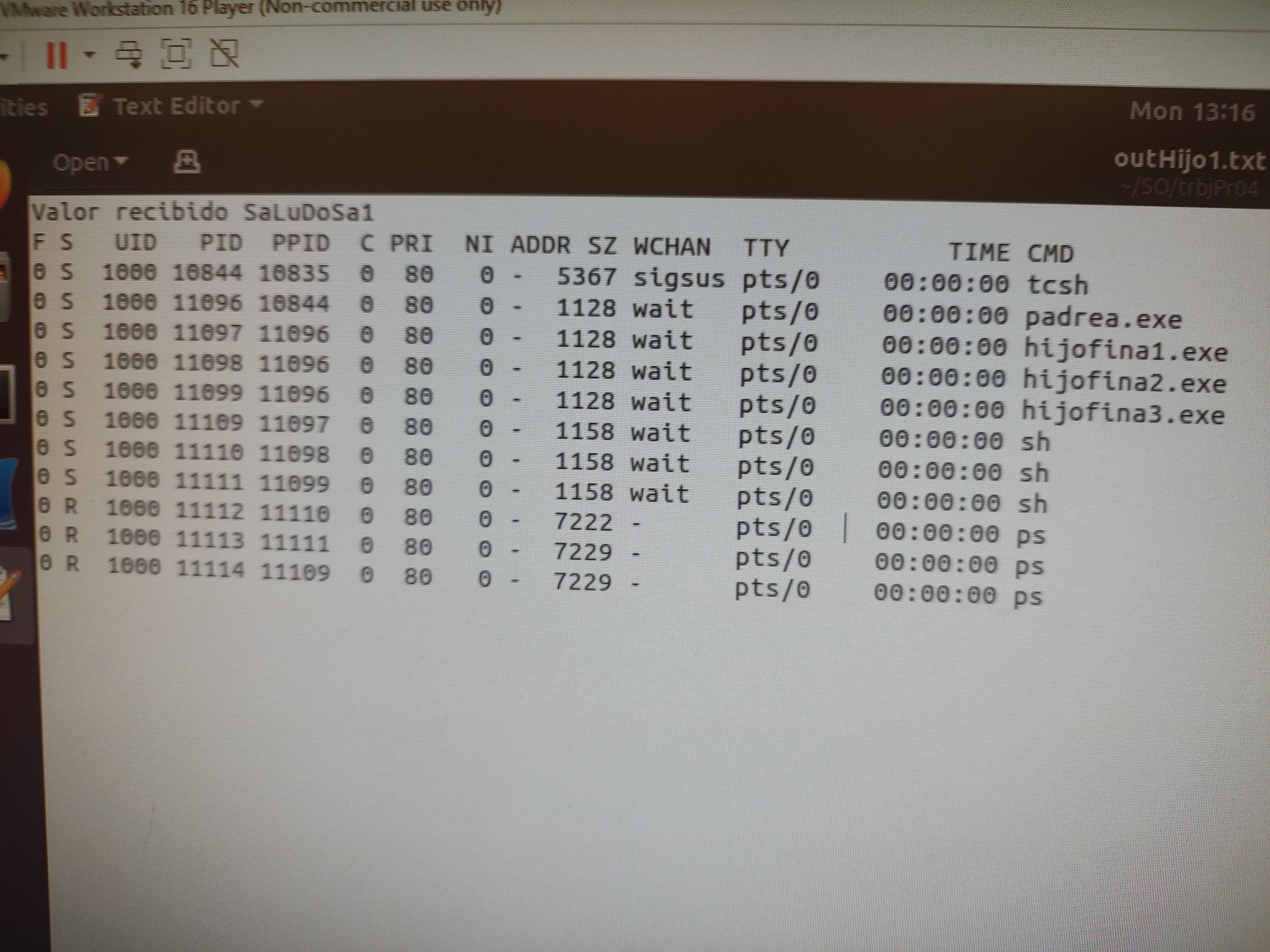
¿En cuál orden terminaron los procesos hijos? **9, 3 y 6. sin embargo podrían terminar en diferente orden al ejecutar de nuevo el programa.**

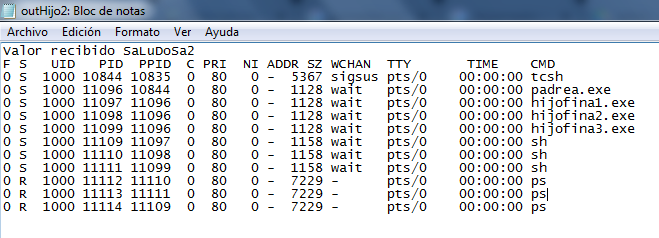
Muestre el despliegue de los tres archivos de texto.

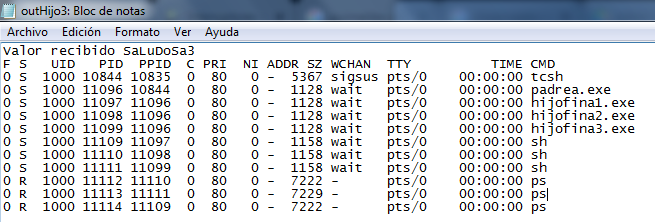
**Los archivos .txt generados se generaron con los siguientes comandos dentro de cada programa “hijofina”, evidentemente se cambiaron los nombres de los archivos de salida para generar 3 archivos distintos de cada hijo.**

****

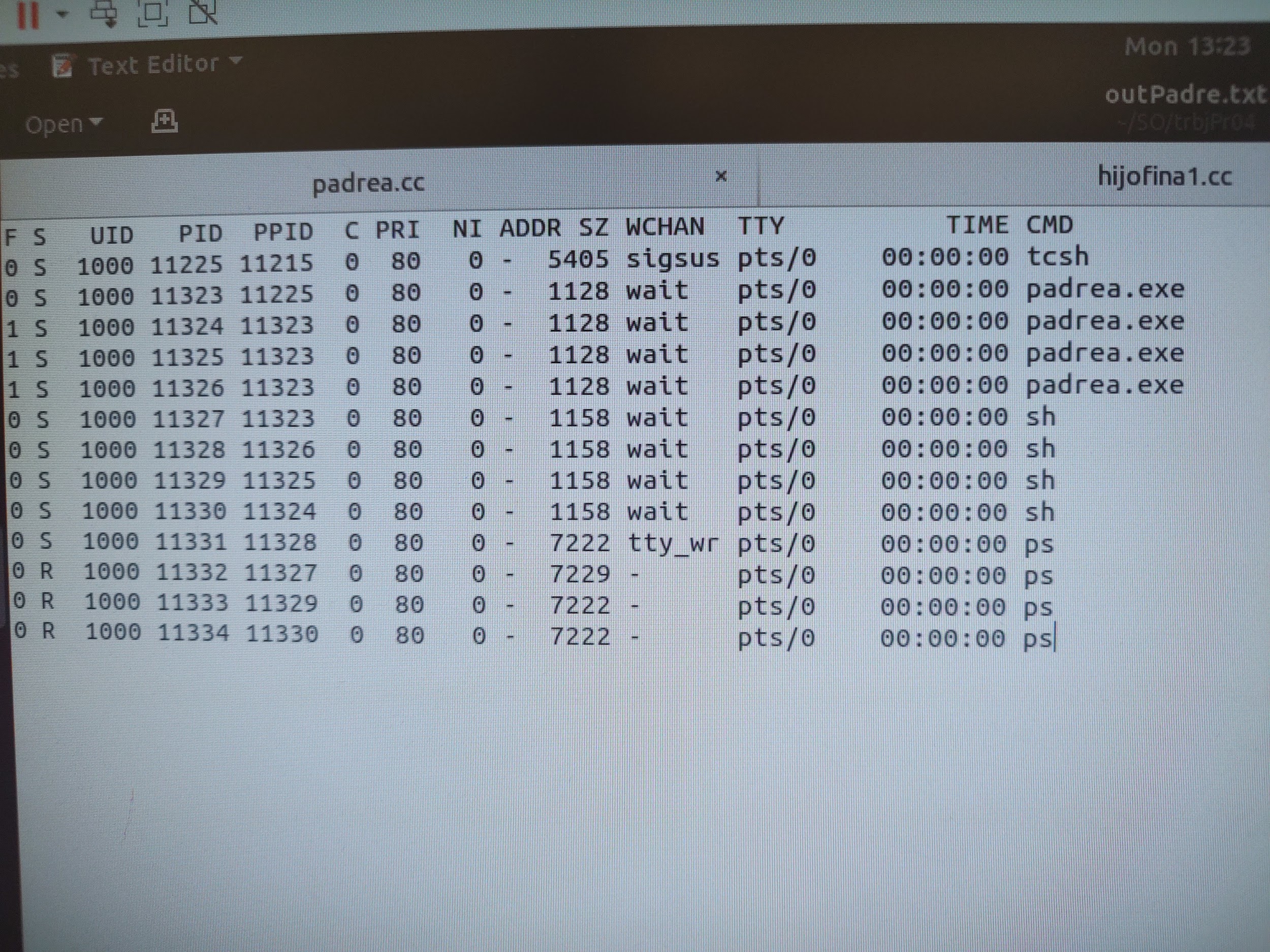
**Y se obtuvieron los archivos .txt siguientes de cada hijo.**







Muestre el árbol de procesos a partir del proceso padre. Diga cuales son los hijos del padre y cuales lo nietos



**Como podemos ver en la foto antes presentada el padre tiene el PID de 11323, 11324, 11325 y 11326, mientras que sus hijos, 11327, 11328, 11329 y 11330 y finalmente los nietos los restantes, 11331, 11332, 11333 y 11334.**

El proceso padre corre en el primer plano. Los procesos hijos y nietos corren en el background. ¿Qué implicación en el uso de recursos tiene este estado de correr en el background?

**Esto indica la jerarquía en la que se van a correr, principalmente está el proceso padre que se va ejecutar en primer plano, y en segundo plano los hijos y nietos. Lo anterior significa una mayor asignación de recursos al padre ya que los procesos en segundo plano se ejecutan con independencia y el proceso padre debe recoger el resultado de la ejecución de los hijos para finalizar adecuadamente, además de que depende (el padre por estar en primer plano) del usuario y su interacción.**

**CONCLUSIÓN**

Al finalizar el trabajo tuvimos la oportunidad de explorar las distintas manipulaciones de procesos, particularmente para la creación de subprocesos hijo y nieto, lo cual nos permitió familiarizarnos más con la teoría vista en clase. También, aprendimos a usar de forma correcta las llamadas a sistemas como exec(), para poder visualizar su funcionamiento. Finalmente, con el uso de distintas variables fuimos capaces de completar la práctica de forma exitosa.