Sistemas Operativos

Práctica 1. Procesos, señales, tuberías, archivos y memoria compartida.

Manuel Morales Amat DNI 48789509T

Grupo Val 17:00 - 19:00 Lunes.

El primer ejercicio de ésta, nuestra práctica, consta de dos partes. La primera parte nos insta a hacer un programa llamado **malla** que genera un una malla de X por Y procesos. Utilizando las llamadas a las funciones **fork**(que genera un nuevo proceso) y **pause**(para la ejecución del código hasta que reciba una señal) podemos generar un árbol de procesos con forma de malla de dimensiones X e Y, generando los nuevos procesos, que llamaremos hijos, y pasandolos después de haber sido padres, que es como llamaremos a aquellos procesos donde se ejecute una fork.
Tras ejecutar nuestro código en la terminal con el comando: **./malla 5 5 &**

```
lolo@NotAPC:~/Escritorio/so/sistemas_operativos/prac1$ ./malla 5 5 &
[1] 7012
```

Ejecutamos la orden **pstree -c -p | grep malla** obteniendo la siguiente salida:

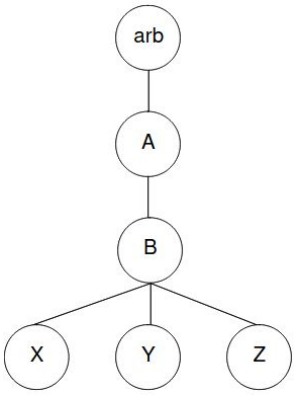
```
lolo@NotAPC:~/Escritorio/so/sistemas_operativos/prac1$ pstree -c -p | grep malla
           |               |-malla(6709)---malla(6712)---malla(6719)---malla(6723)---malla(6729)
           |               |-malla(6710)---malla(6714)---malla(6721)---malla(6725)---malla(6727)
           |               |-malla(6711)---malla(6716)---malla(6720)---malla(6732)---malla(6733)
           |               |-malla(6713)---malla(6718)---malla(6722)---malla(6726)---malla(6730)
           |               |-malla(6715)---malla(6717)---malla(6724)---malla(6728)---malla(6731)
           |               |-malla(6784)---malla(6787)---malla(6793)---malla(6795)---malla(6798)
           |               |-malla(6785)---malla(6789)---malla(6792)---malla(6797)---malla(6801)
           |               |-malla(6786)---malla(6791)---malla(6796)---malla(6799)---malla(6802)
           |               |-malla(6788)---malla(6794)---malla(6800)---malla(6803)---malla(6804)
           |               |-malla(6790)---malla(6805)---malla(6806)---malla(6807)---malla(6808)
           |               |-malla(7013)---malla(7016)---malla(7023)---malla(7026)---malla(7029)
           |               |-malla(7014)---malla(7018)---malla(7025)---malla(7028)---malla(7030)
           |               |-malla(7015)---malla(7020)---malla(7033)---malla(7035)---malla(7036)
           |               |-malla(7017)---malla(7021)---malla(7024)---malla(7027)---malla(7031)
           |               |-malla(7019)---malla(7022)---malla(7032)---malla(7034)---malla(7037)
[1]+  Hecho                   ./malla 5 5
```

También podemos utilizar solamente **pstree -c** que sacara todo el árbol de procesos ejecutados por el sistema, en nuestro caso systemd:

```
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
├─malla───malla───malla───malla───malla
```

De esta manera podemos observar que nuestro programa funciona de manera correcta.

La segunda parte del ejercicio nos pide hacer otro programa que se llamará **ejec.** Este programa debe generar un árbol de procesos con la siguiente forma:



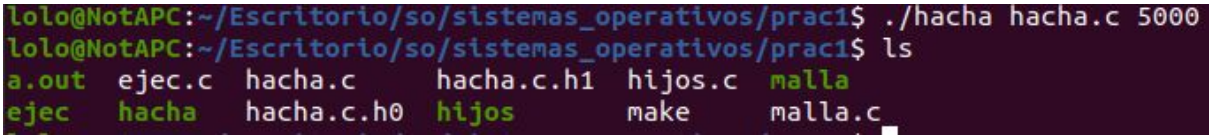
Para ello, repetimos la técnica del apartado anterior, utilizamos **fork** para generar el hijo A, y **pause** para parar al padre, de momento; y así hasta llegar a Z.
Para lograr el comportamiento deseado, creamos una función llama *alarma* de tipo void que no reciba argumentos, en su interior puede estar vacía o podemos ejecutar cualquier cosa dentro; llegado al hijo Z utilizamos la función **signal** que prepara una señal para que lanze la función *alarma* cuando toque. A continuación, llamamos a la función **alarm** que recibirá por argumentos una cantidad de segundos a esperar, pasados esos segundos activa la señal **SIGALRM** que llamará a la función *alarma.* Después de llamar a esta función el hijo Z debe crear otro hijo que se sacrifique por él al ejecutar la llama de la función **execlp**, ya que esta rutina sustituye todo el código y la imagen en memoria por el comando a ejecutar, ya sea un nuevo programa o una llama al sistema. Tras esto, Z aguarda la salida de **execlp** y cuando ello ocurre Z imprime su mensaje y muere, haciendo qué Y muera, que muera X... y así hasta que muere todo el hilo de procesos generados.
Este programa genera la siguiente salida:

```
lolo@NotAPC:~/Escritorio/so/sistemas_operativos/prac1$ ./ejec 5
Soy el proceso ejec: mi pid es 7776
Soy el proceso A: mi pid es 7777. Mi padre es 7776
Soy el proceso B: mi pid es 7778. Mi padre es 7777. Mi abuelo es 7776
Soy el proceso X: mi pid es 7779. Mi padre es 7778. Mi abuelo es 7777. Mi bisabuelo es 7776
Soy el proceso Y: mi pid es 7780. Mi padre es 7778. Mi abuelo es 7777. Mi bisabuelo es 7776
Soy el proceso Z: mi pid es 7781. Mi padre es 7778. Mi abuelo es 7777. Mi bisabuelo es 7776
ejec(7776)──ejec(7777)──ejec(7778)─┬─ejec(7779)
                                   ├─ejec(7780)
                                   └─ejec(7781)──pstree(7782)
Soy Z (7781) y muero
Soy Y (7780) y muero
Soy X (7779) y muero
Soy B (7778) y muero
Soy A (7777) y muero
Soy ejec (7776) y muero
```

En este caso la ejecución de **ejec** ha tardado 5 segundos ya que es el parámetro que se le ha pasado como tiempo de espera para la alarma.

El segundo ejercicio, más enfocado al manejo de archivos y tuberías, nos encomienda la tarea de generar nuevos archivos que contengan secciones de otro archivo más grande. Como argumentos, **hacha** (que es como se pide nombrar al archivo) recibirá el nombre del archivo a dividir, y el tamaño de los bloques en los que se va seccionar.
Para ello, abrimos el archivo con la función **open** la cual devuelve la posición del archivo como un entero. Mediante el uso de la función **lseek** posicionamos el puntero de lectura al principio y guardamos la posición, y repetimos pero moviendo el puntero a la posición final del archivo; restando la posición final menos la inicial obtenemos el tamaño del archivo. Dividiendo el tamaño del archivo entre el tamaño de los bloques encontramos el número de hijos que habrá que generar, siempre generamos 1 hijo más para evitar que no se copie todo el documento, siempre es preferible borrar un archivo vacío que perder información. Sabiendo esto, generamos un array de tantos hijos como necesitemos por 2, quedando una matriz de X hijos por 2. Pongamos que necesitamos 4 hijos, entonces generamos un array de 4x2. Utilizando el método **pipe** cargamos en nuestro array los dos descriptores de las tuberías. Volviendo a usar **lseek** y desplazandonos el número de hijos generados por el tamaño de los bloques, vamos leyendo el archivo y lo vamos escribiendo seccionado en las tuberías de cada hijo. Una vez hecho esto, cada hijo lee con **read** de su tubería la sección de archivo que le corresponda y escribe con **write** en un archivo con el mismo nombre que el pasado por parámetro pero seguido por '.h' y el número de hijo.

Si ejecutamos el hacha, sobre el propio código del hacha obtenemos:



El último de los ejercicios nos pide crear un árbol de procesos con forma muy similar al del apartado b del ejercicio 1.
Esta vez, utilizaremos las funciones **shmget**(para reservar memoria) y **shmat**(para obtener la dirección de la memoria reservada).
Haremos 2 punteros de tipo **pid_t** o **int**, pues son análogos, y haremos que dichos punteros tengan la dirección de memoria retornada por **shmat.**
Cada vez que se genere un hijo guardaremos su PID en uno de estos dos punteros. Dado que generamos 2 punteros, podemos discernir muy fácilmente si se trata de hijos o de hermanos.
Una vez que llegue a los nietos finales, hacemos que estos impriman todos los PIDs de sus padres almacenados en uno de los punteros y mueran. Cuando hayan muerto todos los descendientes del proceso original, éste leerá en la memoria todos los PIDs de los descendientes caídos y los imprimirá en pantalla.
Al ejecutar nuestro código obtendremos esta salida: