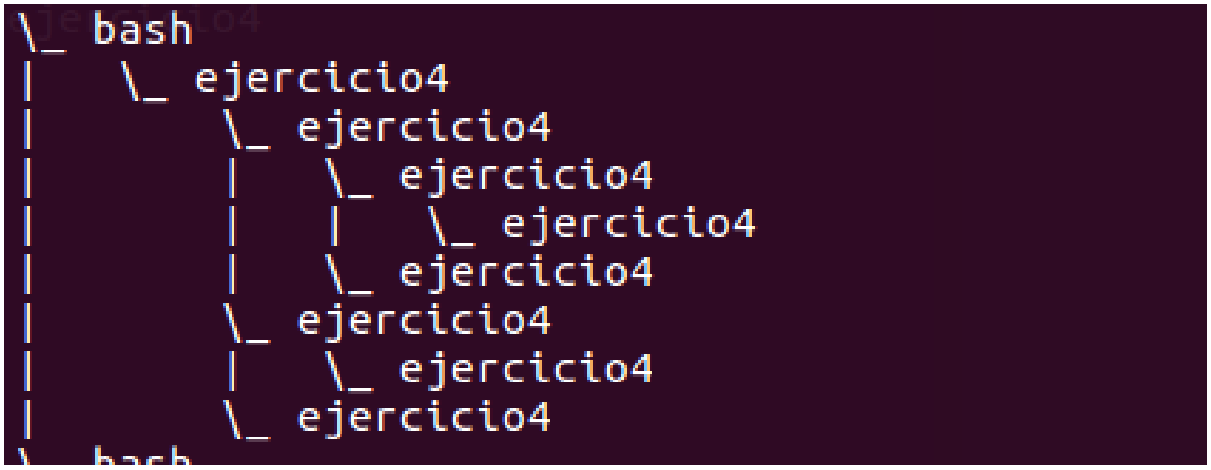
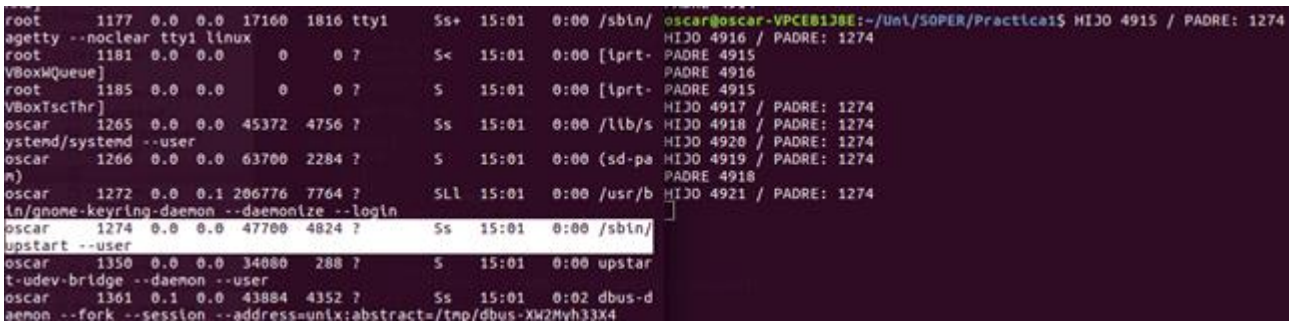
MEMORIA PRACTICA 1 SISTEMAS OPERATIVOS 2016-2017

Ejercicio 4:

Podemos apreciar que tenemos un padre con 3 hijos, cada uno dividiéndose de forma triangular de manera que el último hijo no tiene descendencia.



En el primer código todos los hijos se quedan huérfanos tal y como se puede apreciar en la siguiente fotografía, vemos que todos los procesos tienen como padre el proceso 1274, que es el upstart:

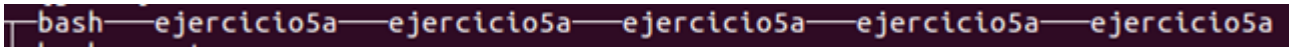


Esto se debe a que no hay ningún wait en el código, aspecto que se implementa en el segundo ejemplo, pero en este caso podemos seguir teniendo huérfanos, porque el primer proceso padre tendrá 3 hijos y solo esperará por uno. Por tanto, podemos tener huérfanos en ambas implementaciones.

Ejercicio 5:

Comenzamos a modificar el código, primero decidimos extraer el fork() del bucle, para que la comprobación en el bucle se realice sobre un dato cambiante. Dentro del bucle sólo realizaremos un fork() tras comprobar que el proceso sea un hijo. De esta forma nos aseguramos que los procesos se creen de manera secuencial.

En la siguiente imagen se puede comprobar que el resultado es el deseado:



En la segunda parte de este ejercicio hacemos uso de un while(wait(NULL)>0), de manera que el padre espera a la finalización de todos los hijos de manera que el resultado es el siguiente:

```
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$ ./ejercicio5b
PADRE 8326
HIJO 8328 / PADRE: 8326
HIJO 8328 / PADRE: 8326
PADRE 8326
PADRE 8326
HIJO 8329 / PADRE: 8326
HIJO 8327 / PADRE: 8326
HIJO 8327 / PADRE: 8326
HIJO 8327 / PADRE: 8326
```

Como podemos ver, todos los procesos hijos provienen del proceso padre y no hay ningún huérfano.

Ejercicio 6:

En el código realizado en este ejercicio hemos decidido introducir “Vacío” en el String reservado para poder comprobar si el String introducido coincide.
Al ejecutarlo vemos que el String del hijo es diferente al String del padre, esto sucede debido a que el String del padre y del hijo son punteros a memoria virtual. Al hacer el fork(), la memoria virtual se copia, haciendo que los punteros no “apunten” a la misma dirección de memoria física.

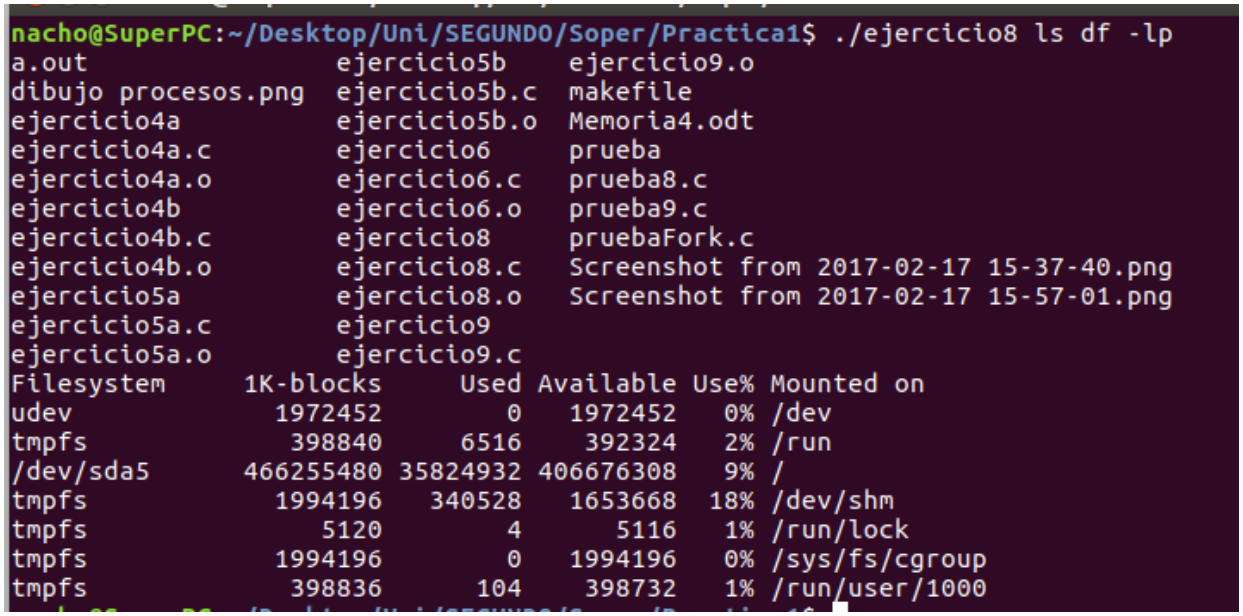
```
nacho@SuperPC:~/Desktop/Uni/SEGUNDO
Hijo
La cadena en el hijo vale: Hijo
La cadena en el padre vale: Vacio
```

Ejercicio 8:

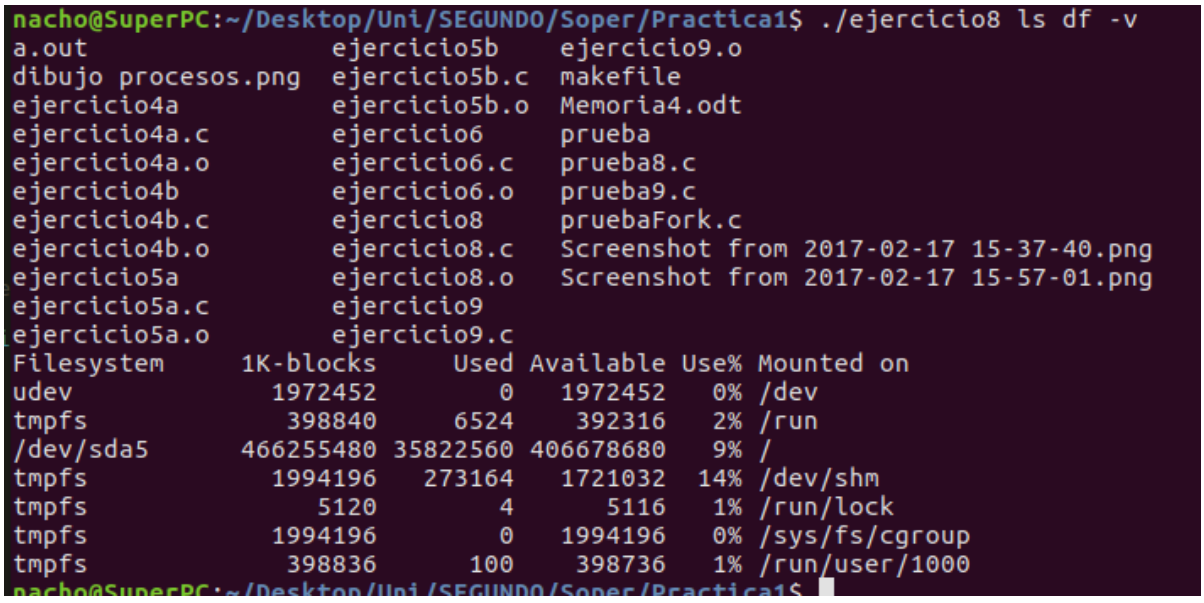
En este ejercicio tratamos las funciones exec, para ello decidimos ejecutar únicamente programas situados en /bin/, por lo que para las rutas absolutas concatenaremos el nombre del programa a dicha ruta.
En el caso del flag -l, ejecutamos la función execl() en un bucle, introduciéndole cada vez el programa correspondiente de los argumentos introducidos al ejecutar el programa.

```
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$ ./ejercicio8 ls df -l
a.out              ejercicio5b    ejercicio9.o
dibujo procesos.png  ejercicio5b.c  makefile
ejercicio4a        ejercicio5b.o  Memoria4.odt
ejercicio4a.c      ejercicio6     prueba
ejercicio4a.o      ejercicio6.c   prueba8.c
ejercicio4b        ejercicio6.o   prueba9.c
ejercicio4b.c      ejercicio8     pruebaFork.c
ejercicio4b.o      ejercicio8.c   Screenshot from 2017-02-17 15-37-40.png
ejercicio5a        ejercicio8.o   Screenshot from 2017-02-17 15-57-01.png
ejercicio5a.c      ejercicio9
ejercicio5a.o      ejercicio9.c
Filesystem     1K-blocks     Used Available Use% Mounted on
udev             1972452        0   1972452   0% /dev
tmpfs             398840     6520    392320   2% /run
/dev/sda5      466255480 35819920 406681320   9% /
tmpfs            1994196   181648   1812548  10% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
tmpfs            1994196        0   1994196   0% /sys/fs/cgroup
tmpfs             398836      100    398736   1% /run/user/1000
```

En el caso del flag -lp, ejecutamos la función execlp() en un bucle, introduciéndole cada vez el programa correspondiente de los argumentos introducidos al ejecutar el programa con su path absoluto.



En el caso del flag -v, ejecutamos la función execv() en un bucle, introduciéndole cada vez el programa correspondiente de los argumentos introducidos al ejecutar el programa. La función execv() permite el paso de varios argumentos, pero tras leer el enunciado hemos entendido que cada hijo debe ejecutar un execv independiente, y por ello lo hemos implementado de esta manera.



En el caso del flag -vp, ejecutamos la función execvp() en un bucle, introduciéndole cada vez el programa correspondiente de los argumentos introducidos al ejecutar el programa con su path absoluto. La función execvp() permite el paso de varios argumentos, pero tras leer el enunciado hemos entendido que cada hijo debe ejecutar un execv independiente, y por ello lo hemos implementado de esta manera.

```
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$ ./ejercicio8 ls df -vp
a.out                ejercicio5b    ejercicio9.o
dibujo procesos.png  ejercicio5b.c  makefile
ejercicio4a          ejercicio5b.o  Memoria4.odt
ejercicio4a.c        ejercicio6     prueba
ejercicio4a.o        ejercicio6.c   prueba8.c
ejercicio4b          ejercicio6.o   prueba9.c
ejercicio4b.c        ejercicio8     pruebaFork.c
ejercicio4b.o        ejercicio8.c   Screenshot from 2017-02-17 15-37-40.png
ejercicio5a          ejercicio8.o   Screenshot from 2017-02-17 15-57-01.png
ejercicio5a.c        ejercicio9
ejercicio5a.o        ejercicio9.c
Filesystem     1K-blocks     Used Available Use% Mounted on
udev             1972452        0   1972452   0% /dev
tmpfs             398840     6524    392316   2% /run
/dev/sda5      466255480 35824140 406677100   9% /
tmpfs            1994196   279268   1714928  15% /dev/shm
tmpfs               5120        4      5116   1% /run/lock
tmpfs            1994196        0   1994196   0% /sys/fs/cgroup
tmpfs             398836      100    398736   1% /run/user/1000
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$
```

Ejercicio 9:

En este ejercicio comenzamos el trabajo con pipes, para ello cada vez que realizamos un fork(), creamos un pipe nuevo, lo que nos permite la comunicación entre padre e hijo.
En un principio, el programa funcionaba correctamente, pero la impresión en el proceso padre imprimía caracteres basura al final del readbuffer. Por ello hemos decidido tokenizar la cadena, obteniendo el resultado deseado:

```
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$ ./ejercicio9
Introduzca dos operandos: 20 40
Datos enviados a través de la tubería por el proceso PID= 10797
Operando 1: 20. Operando 2: 40. Suma: 60
Datos enviados a través de la tubería por el proceso PID= 10798
Operando 1: 20. Operando 2: 40. Resta: -20
Datos enviados a través de la tubería por el proceso PID= 10799
Operando 1: 20. Operando 2: 40. Producto: 800
Datos enviados a través de la tubería por el proceso PID= 10800
Operando 1: 20. Operando 2: 40. Cociente: 0
nacho@SuperPC:~/Desktop/Uni/SEGUNDO/Soper/Practica1$
```