



# SISTEMAS OPERATIVOS

## *Practica 2: Gestión de Procesos*

Carlos Contreras Sanz 100303562  
Álvaro Gómez Ramos 100307009

# Contenido

Ejercicio 0 ..... 3

Ejercicio 1 ..... 4

Ejercicio 2 ..... 4

Ejercicio 3 ..... 5

Ejercicio 4 ..... 5

Ejercicio 5 ..... 6

Ejercicio 6 ..... 7

Ejercicio 7 ..... 7

Ejercicio 8 ..... 8

Ejercicio 9 ..... 9

Comprobar formato ..... 9

Conclusiones ..... 9

En esta práctica se pretendía hacernos ver cómo se lleva a cabo la gestión de procesos, la comunicación entre ellos y con el sistema operativo.

Hemos desarrollado por tanto, un minishell que ejecuta una serie de comandos a elección del usuario, y que tan solo se detendrá en caso de que elijamos la opción de exit.

Para ello hemos usado un bucle “infinito” (prácticamente) que se ejecutara una y otra vez, leyendo de teclado el comando que introduzcamos, y en función de eso realizara la acción deseada. Guardamos lo que leemos del teclado en un String que será lo que compararemos para saber en qué situación estamos. Además, en cualquier momento se capturara la señal de Ctrl+Z, aunque de esto ya hablaremos en el ejercicio 9.

Pasamos por tanto a cada uno de los ejercicios (diferentes comandos), que serán reconocidos dentro del while.

En todos los casos, al crear hijos, abrir ficheros y similares, lo primero que hemos hecho es comprobar el correcto funcionamiento de esas instrucciones.

Además, siempre, el padre permanece inalterable, y todos los cambios se realizan sobre los hijos, y no solo eso, si no que salvo en el ejercicio 8, espera a la terminación de sus hijos para seguir ejecutando, y volver a pedir la instrucción de la minishell (esto se ve en que no se imprime “minishell” antes de la salida de la instrucción anterior).

## Ejercicio 0

Aquí se desarrolla tan solo un bucle infinito, con un while. Hemos optado por incluir una condición de un int desigual de -1, con la idea inicial de usar esa variable para, en caso de que hubiese algún error, cambiar el valor a -1 y salir del bucle, pero finalmente lo que hemos hecho es hacer returns 0 o -1, en función de la ocasión, de forma que sale del main y de la ejecución, sin necesidad de hacer uso de la variable de parada.

Esto quiere decir que no es realmente infinito el bucle, pero dado el rango de los int, hasta que se dé el caso de llegar a -1, habrá habido un numero enorme de ejecuciones, haciéndolo virtualmente infinito (nadie es capaz de insertar a mano tantas instrucciones).

Se añaden dentro del bucle, tantas comprobaciones if/else if como comandos tengamos.

Para este primer ejercicio se añade la comprobación de que la cadena leída por teclado sea igual a “exit”, con lo que se hará un return 0 y se saldrá de la ejecución del programa.

Comprobamos que sale correctamente de la ejecución del minishell cuando se lo pedimos:

```
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ ./minishell
minishell>command1
 12:05:54 up 31 min,  2 users,  load average: 0.18, 0.26, 0.28
minishell>exit
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$
```

## Ejercicio 1

En este ejercicio se nos pedía ejecutar el comando “uptime” al introducir command1 en la minishell.

Para ello, una vez comprobado que se trata de este comando el que queremos ejecutar, se crea un hijo mediante el fork (para que no se vea alterado el padre) y mediante execvp se cambia ese hijo por la ejecución de la instrucción que nos piden. Como siempre, el padre (while) no se modifica y espera al hijo para iterar otra vez.

Para comprobar el correcto funcionamiento de este ejercicio, basta con comparar el resultado de introducir “command1” con el resultado de la ejecución del comando uptime en la terminal de Linux.

```
minishell>command1
11:44:49 up 10 min, 3 users, load average: 0.10, 0.38, 0.30
minishell>
```

```
xinzodl@XinZodl-Ubuntu: ~
xinzodl@XinZodl-Ubuntu:~$ uptime
11:44:52 up 10 min, 3 users, load average: 0.09, 0.37, 0.30
xinzodl@XinZodl-Ubuntu:~$
```

## Ejercicio 2

En este ejercicio, queremos que al introducir “command2” en la minishell, ejecute la instrucción “uname -a” de Linux.

La única diferencia con la ejecución del ejercicio anterior es, aparte de la instrucción en sí misma, el hecho de estar compuesta la instrucción por dos parámetros, que se pasarán al execvp en posiciones diferentes de un String.

Al igual que en el caso anterior, con comparar la salida de la instrucción de la terminal con la que da la minishell, sabremos si funciona o no.

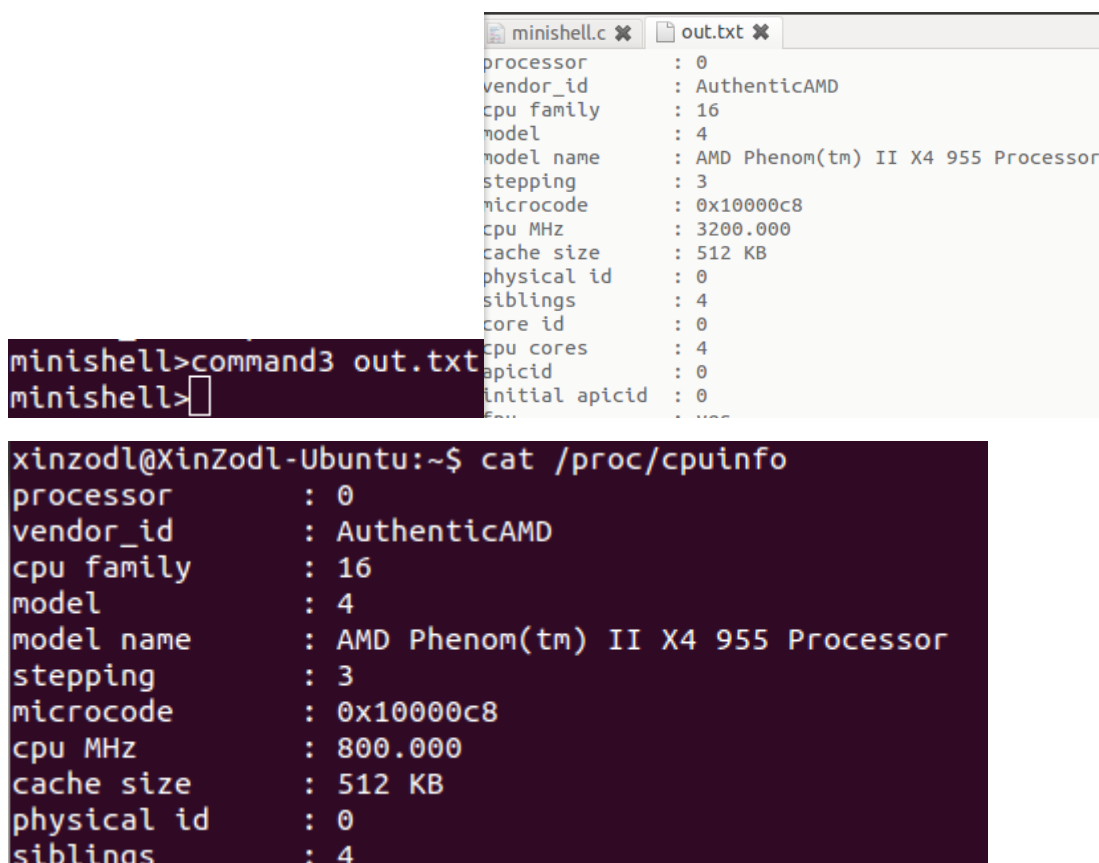
```
minishell>command2
Linux XinZodl-Ubuntu 3.8.0-37-generic #53~precise1-Ubuntu SMP Wed Feb 19 21:37:54 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
minishell>
```

```
xinzodl@XinZodl-Ubuntu:~$ uname -a
Linux XinZodl-Ubuntu 3.8.0-37-generic #53~precise1-Ubuntu SMP Wed Feb 19 21:37:54 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
xinzodl@XinZodl-Ubuntu:~$
```

## Ejercicio 3

En este caso hemos creado un hijo a partir del padre. Primero hemos redirigido la salida estándar de ese proceso hijo a un archivo que nos pasan como parámetro (que se lee del teclado de la misma forma que los comandos, y se comprueba que se puede abrir o crear), y después se ejecuta el `execvp` de la misma forma que en los casos anteriores, con la diferencia de la instrucción en este caso.

Para comprobar si funciona, basta con asegurarnos de que la salida del fichero es la misma que la que daría el comando escrito en la terminal.



The image shows a terminal window with two tabs: `minishell.c` and `out.txt`. The `minishell.c` tab shows the following output:

```
processor      : 0
vendor_id     : AuthenticAMD
cpu family    : 16
model         : 4
model name    : AMD Phenom(tm) II X4 955 Processor
stepping      : 3
microcode     : 0x10000c8
cpu MHz       : 3200.000
cache size    : 512 KB
physical id   : 0
siblings      : 4
core id       : 0
cpu cores     : 4
apicid        : 0
initial apicid : 0
```

The `out.txt` tab shows the same output as the `minishell.c` tab.

Below the terminal window, a command prompt shows the execution of the `cat /proc/cpuinfo` command, which produces the same output as the `minishell.c` tab:

```
xinzodl@XinZodl-Ubuntu:~$ cat /proc/cpuinfo
processor      : 0
vendor_id     : AuthenticAMD
cpu family    : 16
model         : 4
model name    : AMD Phenom(tm) II X4 955 Processor
stepping      : 3
microcode     : 0x10000c8
cpu MHz       : 800.000
cache size    : 512 KB
physical id   : 0
siblings      : 4
```

## Ejercicio 4

Ahora vamos a realizar un `grep` sobre un fichero de texto (el del ejercicio anterior).

Por tanto lo que hemos hecho es cambiar la entrada estándar del hijo, al fichero y ejecutar el `execvp` con las instrucciones correspondientes.

Para comprobarlo, una vez más, nos aseguramos de que la salida de este comando, sea la misma que la de `"cat /proc/cpuinfo | grep "model name" "`.

```
minishell>command4 out.txt
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
minishell>
```

```
xinzodl@XinZodl-Ubuntu:~$ cat /proc/cpuinfo | grep "model name"
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
xinzodl@XinZodl-Ubuntu:~$
```

## Ejercicio 5

Ahora, ejecutaremos el comando “cat /proc/cpuinfo | grep “model name” ” al escribir command5 en la minishell.

Necesitamos dos hijos, creados a partir del padre, y una tubería. Uno de los hijos, se encargara de la primera parte el comando y redirigirá su salida a la tubería, mientras que el otro se encargara de la segunda parte y lo que redirigirá será su entrada.

La comprobación una vez más para asegurarnos de su correcto funcionamiento vamos a comparar lo que sale al ejecutar el comando en la terminal, y el command5 en la minishell.

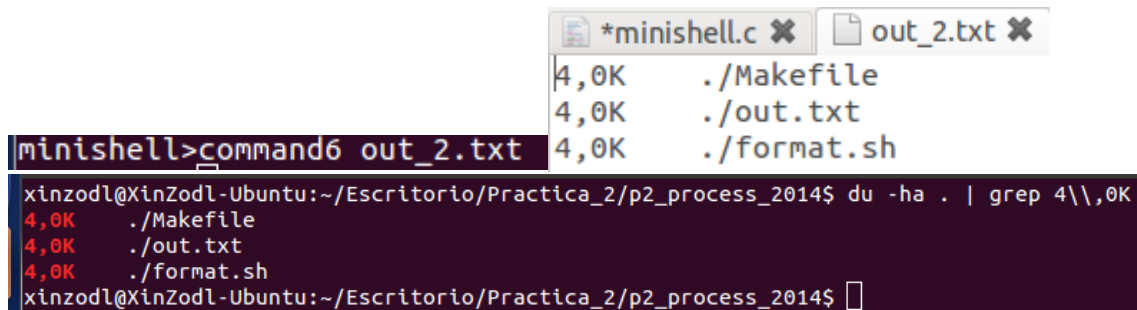
```
minishell>command5
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
minishell>
```

```
xinzodl@XinZodl-Ubuntu:~$ cat /proc/cpuinfo | grep "model name"
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
model name      : AMD Phenom(tm) II X4 955 Processor
xinzodl@XinZodl-Ubuntu:~$
```

## Ejercicio 6

Este caso es similar al anterior (con el detalle de que el comando es diferente), pero en lugar de mostrar el resultado por pantalla, se mostrara en un archivo. Por ello, realizamos los mismos pasos que en el ejercicio anterior, cambiando además la salida del segundo proceso, al archivo de salida que nos facilitan.

Para comprobar el funcionamiento, volvemos a comparar la salida del command6 con lo que saldría en la terminal con el mismo comando.



```
minishell>command6 out_2.txt
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ du -ha . | grep 4\\,0K
4,0K    ./Makefile
4,0K    ./out.txt
4,0K    ./format.sh
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$
```

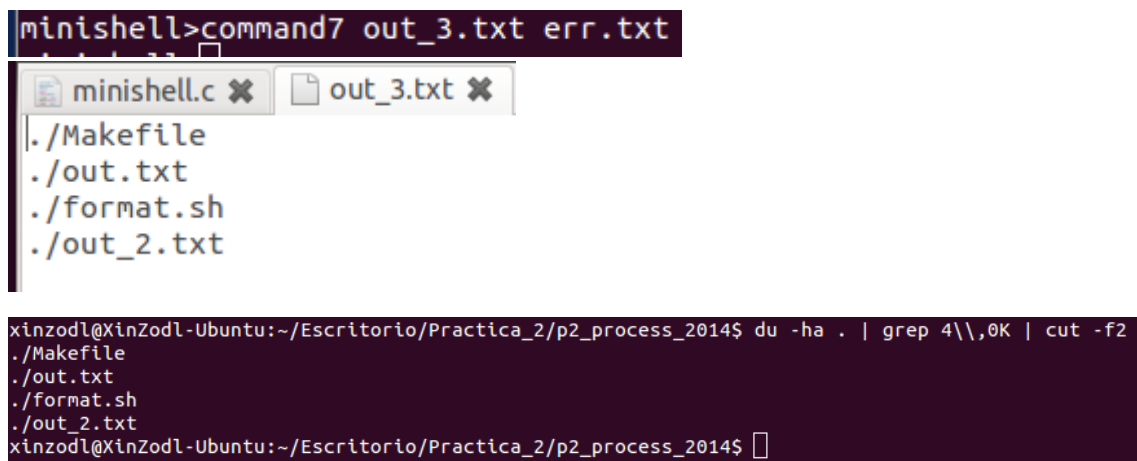
## Ejercicio 7

Ahora lo que se persigue es unir tres procesos (hijos, creados a partir de un único padre) por medio de tuberías, y redireccionar la salida estándar del último, y la salida de error.

Para ello hacemos que el padre cree tres hijos, y las dos tuberías. Con ello cada hijo, puede acceder a la tubería que le convenga y redireccionar lo que necesite (la salida en el caso del primero, salida y entrada en el caso del segundo, a diferentes tuberías, y la entrada del tercero). Además el ultimo hijo redireccionar su salida estándar a un archivo.

En todo caso, y como en todos los ejercicios se tiene muy presente el haber cerrado correctamente las tuberías y demás herramientas usadas, siendo digno de mención en este caso por tener dos tuberías y tres archivos, combinado con redireccionamiento de salida.

Para comprobar este comando, una vez comparamos la salida, con la que daría en la terminal.



```
minishell>command7 out_3.txt err.txt
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ du -ha . | grep 4\\,0K | cut -f2
./Makefile
./out.txt
./format.sh
./out_2.txt
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$
```

Y por otro lado, comprobamos la salida de error, introduciendo en el hijo que ejecuta el `du -ha`, un error.

```
minishell.c x out_3.txt x err.txt x
probando error en du -ha: Success
```

## Ejercicio 8

En este caso lo que hemos hecho es lo mismo que en el ejercicio 6, pero eliminando la parte del código del padre, que espera a la finalización de los hijos, y añadiendo el imprimir el PID del hijo.

Para comprobarlo, hemos ejecutado el comando, y con su PID, lo hemos buscado, apareciendo efectivamente como Z.

```
minishell>command8
out_4.txt
[3595]
minishell>

xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ ps -aux | grep 3595
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
xinzodl  3595  0.0  0.0      0   0 pts/1    Z+   12:01   0:00 [grep] <defunct>
xinzodl  3648  0.0  0.0  13636  892 pts/2    R+   12:02   0:00 grep --color=auto 3595
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$
```

Además, comprobamos el hecho de que si se ejecuta otro comando, el padre sí que recoge el proceso que había dejado en segundo plano.

```
minishell>command8
out_4.txt
[3595]
minishell>command1
12:02:22 up 27 min,  3 users,  load average: 0.67, 0.38, 0.31
minishell>

xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ ps -aux | grep 3595
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
xinzodl  3670  0.0  0.0  13632  892 pts/2    S+   12:02   0:00 grep --color=auto 3595
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$
```



## Ejercicio 9

Ahora se va a abordar el tratamiento de señales.

Vamos a capturar la señal de Ctrl+Z con signal (SIGTSTP, función); y la definición de la función, como una impresión pro pantalla indicando que se está ejecutando.

El signal para capturar la señal, se encuentra dentro del while, para poder usarlo más de una vez.

Para ver si funciona, añadimos un sleep (10) al commando1, por ejemplo, y en ese tiempo, hacemos Ctrl+Z, para asegurarnos de que funciona. Podemos además hacerlo varias veces seguidas y ver que no hay problemas.

```
xinzodl@XinZodl-Ubuntu:~/Escritorio/Practica_2/p2_process_2014$ ./minishell
minishell>command1
^Z
Executing comando: command1
 12:04:04 up 29 min,  3 users,  load average: 0.17, 0.30, 0.29
minishell>command1
^Z
Executing comando: command1
 12:04:07 up 29 min,  3 users,  load average: 0.15, 0.29, 0.29
minishell>command1
 12:04:21 up 29 min,  3 users,  load average: 0.12, 0.28, 0.28
minishell>command2
Linux XinZodl-Ubuntu 3.8.0-37-generic #53~precise1-Ubuntu SMP Wed Feb 19 21:37:54 UTC 2014 x86_64 x86_
64 x86_64 GNU/Linux
minishell>command1
^Z
Executing comando: command1
 12:04:29 up 29 min,  3 users,  load average: 0.11, 0.27, 0.28
minishell>
```

## Comprobar formato

Como ultima comprobación, señalar que llevamos a cabo una comprobación de formato, para asegurarnos de seguir las indicaciones dadas por los profesores.

```
xinzodl@XinZodl-Ubuntu:~/Escritorio$ sh format.sh ssou_p2_100303562_100307009.zip
Correct file name/format
Compilation OK
xinzodl@XinZodl-Ubuntu:~/Escritorio$
```

## Conclusiones

Para terminar, decir que se agradece que esta práctica sea tan gradual. La hace más fácil de seguir y entender, lo que no quiere decir que no hayamos tenidos más o menos problemas para hacerla. Por ejemplo, tuvimos algo de dificultad con cuando y donde cerrar las tuberías descriptores y similares, ya que tan malo es no cerrar algo, como cerrar algo ya cerrado.

Por lo demás, como hemos dicho, el hecho de que sean escalonados los ejercicios, hace que entender uno sencillo, te posibilite hacer uno un poco más difícil sin demasiado trabajo.

Creemos que hemos aprendido bastante con esta práctica, y que nos ha servido, y servirá de utilidad para asentar conocimientos y conceptos.