



SISTEMAS OPERATIVOS

Práctica 2 Llamadas al sistema

Profesor: DANTE DOMIZZI SÁNCHEZ GALLEGOS

CATHERINE ALESSANDRA TORRES CHARLES

Grupo 81

Manuel Manchado Barquero NIA: 100522228

Alejandro Fernández Sánchez NIA: 100522269

Índice:

1. Descripción del código.....	2
1.1. Análisis y descripción función “Scripter”	2
1.2. Análisis y descripción función “Mygrep”	4
2. Batería de pruebas.....	5
2.1. Batería de pruebas función “Scripter”	5
2.2. Batería de pruebas función “Mygrep”	8
3. Conclusión.....	10

1. Descripción del código

1.1. Análisis y descripción función “Scripter”

El primer programa a desarrollar en la segunda práctica es un intérprete de comandos de Linux hecho en C. Debe leer los comandos uno a uno dentro de un script que debe comenzar con la frase `### Script de SSOO`.

Para poder analizar cada comando ya tenemos las funciones necesarias, solamente es necesario programar el procesamiento de los comandos. Debe de procesar cada comando de forma individual creando un hijo que lo ejecute de forma que tenga en cuenta los atributos y condiciones correspondientes.

Hemos decidido dividir el código de la función `main()` en dos partes:

- **Reconocimiento de la línea obligatoria de comienzo:** Se trata de un bloque que se encarga de reconocer la primera línea del script.
- **Lectura y mandado a procesar de cada línea:** El bloque de código a continuación se trata de dos bucles anidados que se encargan de leer y guardar los caracteres de una línea en un string.

El bucle pequeño es capaz de reconocer los saltos de línea de forma que cuando esto ocurra, se acabe el bucle y se mande a procesar la línea completa. De la misma manera puede reconocer líneas en blanco y el final del script de forma que cambia el valor de la variable que mantiene el bucle más grande ejecutándose.

El bucle grande contiene al pequeño y se encarga de mandar las diferentes líneas a procesar cada vez que el bucle menor se acaba.

Es importante recalcar que hemos decidido añadir una función que se llama dentro del `main`, encargada de recoger los hijos que ejecuten los comandos en los que el background esté activo. De esta forma nos aseguramos de que no quede ningún proceso zombie.

En lo referente a la función de procesamiento de líneas, el código que hemos añadido nosotros se ubica dentro del bucle que realiza una iteración por comandos de la línea a procesar.

Al comenzar el bucle lo primero que realizamos es eliminar las comillas que bajan el porcentaje de las pruebas sobre nuestro código.

Al momento creamos una tubería en el caso en el que el comando a ejecutar no sea el último de la línea. Nada más crear la tubería se crea el hijo que realizará el comando y se le asignan la entrada y salida correspondientes:

- Si no se trata del primer comando, la entrada estándar es la lectura de la tubería creada en la iteración anterior.
- Si no se trata del último comando, la salida estándar es la escritura en la tubería creada en la iteración correspondiente al comando.

Trás las redirecciones que se deben realizar de forma obligatoria para conectar los comandos de una misma línea, nos dedicamos a realizar las redirecciones explícitas que se indican en la propia línea de forma que:

- El primer proceso hijo no puede tener redirección de salida
- El último proceso hijo no puede tener redirección de entrada
- Todos los procesos hijos intermedios solo pueden tener redirección de salida de errores

Una vez realizadas todas las redirecciones, se manda al hijo a ejecutar el comando correspondiente.

A continuación, el padre en todas las iteraciones menos en la primera, se encarga de cerrar la tubería de la iteración anterior al haber sido ya utilizada para leer los datos. Cierra la parte de escritura de la tubería actual y guarda el descriptor de la lectura para poder utilizarlo en la siguiente iteración si es necesario. De esta forma conseguimos que el código sea válido para líneas compuestas por n comandos.

Antes de terminar el bucle, si el background está activo, se imprime el identificador de proceso del hijo encargado de ejecutar el último comando de la línea. Siempre que llamamos a una función comprobamos que se realice de forma correcta y si esto no ocurre, que devuelva -1 y se imprime por la salida de error una frase que explique lo sucedido.

1.2. Análisis y descripción función “Mygrep”

El segundo programa a desarrollar dentro de la práctica es un programa que sustituye al grep del shell. Pasándole al programa la dirección del archivo y la cadena de texto a buscar este debe imprimir las líneas en las que encuentre la cadena y si no la encuentra un mensaje de que no la ha encontrado.

Al comenzar la función main realizamos una comprobación de que el número de argumentos de la función sea el esperado, sino se devuelve el error y no se ejecuta más código. Realizamos la apertura del fichero de entrada y también comprobamos que haga de forma correcta sin errores.

Una vez preparadas las variables comenzamos con el bucle principal de la función que se encarga de leer el fichero mientras guarda en un buffer las líneas para poder posteriormente comparar y averiguar si la cadena de caracteres que buscamos se encuentra en dicha línea.

Una vez encontremos la cadena que buscamos, se imprime por pantalla la línea en la que se encuentra dicha cadena (esto se hará hasta que se acabe de leer el fichero). A continuación, se cierran los descriptores y se termina el código con código de finalización 0 si todo se ha ejecutado de forma correcta.

2. Batería de pruebas

2.1. Batería de pruebas función “Scripter”

Test 1: En el primer test queremos cubrir un caso válido con una entrada estándar como la siguiente:

```
## Script de SSOO
ls -l
cat fich | grep palabra
cat fich > fich2 &
ls -l | sort < fichero
```

En el fichero fich está escrito:
“Este fichero es una \n prueba.”

El resultado de la prueba en la terminal es el siguiente:

```
están finalizada$ ./scripter prueba
total 80
-rw-rw-rw- 1 manuel manuel 75 abr 2 00:42 autores.txt
-rw-rw-r-- 1 manuel manuel 27 abr 2 22:38 fich
-rw-r--r-- 1 manuel manuel 27 abr 2 22:39 fich2
-rw-rw-r-- 1 manuel manuel 552 abr 2 22:39 fichero
-rw-rw-rw- 1 manuel manuel 405 abr 2 00:42 Makefile
-rwxrwxr-x 1 manuel manuel 16352 abr 2 22:35 mygrep
-rw-rw-rw- 1 manuel manuel 1748 abr 2 00:42 mygrep.c
-rw-rw-r-- 1 manuel manuel 89 abr 2 22:38 prueba
-rwxrwxr-x 1 manuel manuel 21128 abr 2 22:40 scripter
-rw-rw-rw- 1 manuel manuel 9197 abr 2 22:40 scripter.c
45059
```

Dentro de fich2 se realiza una copia de fich y dentro de fichero se escribe lo siguiente:

```
-rw-r--r-- 1 manuel manuel 27 abr 2 22:40 fich2
-rw-rw-r-- 1 manuel manuel 0 abr 2 22:40 fichero
-rw-rw-r-- 1 manuel manuel 27 abr 2 22:38 fich
-rw-rw-r-- 1 manuel manuel 89 abr 2 22:38 prueba
-rw-rw-rw- 1 manuel manuel 75 abr 2 00:42 autores.txt
-rw-rw-rw- 1 manuel manuel 405 abr 2 00:42 Makefile
-rw-rw-rw- 1 manuel manuel 1748 abr 2 00:42 mygrep.c
-rw-rw-rw- 1 manuel manuel 9197 abr 2 22:40 scripter.c
-rwxrwxr-x 1 manuel manuel 16352 abr 2 22:35 mygrep
-rwxrwxr-x 1 manuel manuel 21128 abr 2 22:40 scripter
total 76
```

Test 2: En el segundo test forzamos el fallo de lectura de la primera frase obligatoria ya que no la hemos escrito. La entrada correspondiente es la siguiente:

```
ls -l
cat fich | grep palabra
cat fich > fich2 &
ls -l | sort < fichero
```

La salida que obtenemos en la prueba es:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Version finalisima$ ./scripter prueba
El fichero comienza erroneamente
: Success
```

Test 3: En el tercer test queremos probar el reconocimiento de líneas en blanco que hacen que se pare el scripter. La entrada es la siguiente:

```
## Script de SSOO
ls -l

cat fich > fich2 &
ls -l | sort < fichero
```

La salida obtenida es la deseada, ya que imprime por pantalla el error que estábamos buscando:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Version finalisima$ ./scripter prueba
total 80
-rw-rw-rw- 1 manuel manuel 75 abr 2 00:42 autores.txt
-rw-rw-r-- 1 manuel manuel 27 abr 2 22:38 fich
-rw-r--r-- 1 manuel manuel 27 abr 2 22:40 fich2
-rw-rw-r-- 1 manuel manuel 552 abr 2 22:40 fichero
-rw-rw-rw- 1 manuel manuel 405 abr 2 00:42 Makefile
-rwxrwxr-x 1 manuel manuel 16352 abr 2 22:35 mygrep
-rw-rw-rw- 1 manuel manuel 1748 abr 2 00:42 mygrep.c
-rw-rw-r-- 1 manuel manuel 90 abr 2 22:45 prueba
-rwxrwxr-x 1 manuel manuel 21128 abr 2 22:40 scripter
-rw-rw-rw- 1 manuel manuel 9197 abr 2 22:40 scripter.c
Línea vacía: No child processes
```

Test 4: En el cuarto test nuestro objetivo es ser capaces de atrapar el error que genera un comando desconocido. La entrada es la siguiente:

```
## Script de SSOO
inventado fich | grep palabra
```

Conseguimos el resultado esperado ya que por pantalla se imprime lo siguiente:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Version finalisima$ ./scripter prueba
Fallo a la hora de ejecutar el hijo.: No such file or directory
```

Test 5: En el quinto test probamos si nuestro código es capaz de procesar más de 5 comandos en una sola línea sin que ocurra ningún error. El código en cuestión es el siguiente:

```
## Script de SSOO
cat fich | grep palabra | ls -l | sort | cat fich > fich2
```

El resultado de la prueba es el esperado, sin ningún error:

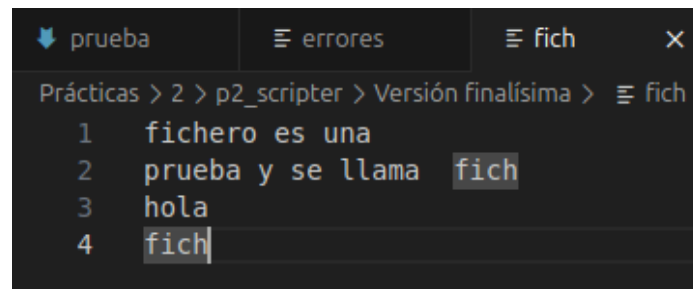
```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Ve
rsión finalísima$ ./scripter prueba
```

En los ficheros fich y fich2 se guarda lo siguiente:

The image shows two side-by-side screenshots of a text editor. The left screenshot shows the 'fich' file with two lines: '1 Este fichero es una' and '2 prueba y se llama fich.' The right screenshot shows the 'fich2' file with the same two lines: '1 Este fichero es una' and '2 prueba y se llama fich.'

2.2. Batería de pruebas función “Mygrep”

Test 1: En el primer test probamos un caso válido sin ningún tipo de error cuyo fichero de lectura contiene la siguiente entrada:

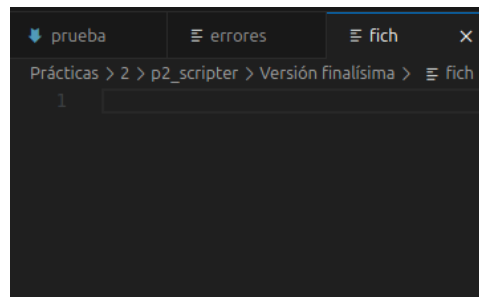


A screenshot of a file editor window titled 'prueba'. It has tabs for 'errores' and 'fich'. The breadcrumb path is 'Prácticas > 2 > p2_scripter > Versión finalísima > fich'. The file 'fich' contains four lines of text: '1 fichero es una', '2 prueba y se llama fich', '3 hola', and '4 fich'.

La salida que obtenemos es la que esperamos. El resultado obtenido es:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Versión finalísima$ ./mygrep fich fich
fichero es una
prueba y se llama fich
fich
```

Test 2: En el segundo test probamos lo que ocurre cuando el fichero de entrada está totalmente vacío:

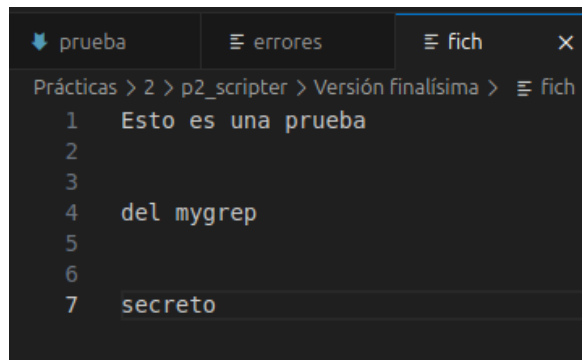


A screenshot of a file editor window titled 'prueba'. It has tabs for 'errores' and 'fich'. The breadcrumb path is 'Prácticas > 2 > p2_scripter > Versión finalísima > fich'. The file 'fich' is empty, with only a line number '1' visible at the top.

El resultado obtenido por terminal es el siguiente:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Versión finalísima$ ./mygrep fich fich
fichero es una
prueba y se llama fich
fich
```

Test 3: En el tercer test queremos probar un caso un poco extremo en el que el fichero esté lleno de espacios en blanco y las palabras están muy dispersas:



```
prueba  errores  fich  X
Prácticas > 2 > p2_scripter > Versión finalísima > fich
1  Esto es una prueba
2
3
4  del mygrep
5
6
7  secreto
```

La salida obtenida es la siguiente:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Ve
rsión finalísima$ ./mygrep fich secreto
secreto
```

Test 4: En el cuarto test probamos lo que ocurre cuando el fichero que se le pasa como valor de entrada es inexistente. El resultado es el siguiente:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Ve
rsión finalísima$ ./mygrep no_existe secreto
Error al abrir el archivo: No such file or directory
```

Test 5: En la quinta prueba nos fijamos en lo que ocurre si los parámetros son más de lo esperado. Al ejecutarlo se imprime por pantalla lo siguiente:

```
manuel@manuel-VirtualBox:~/Documentos/SS00/ProblemasC/Prácticas/2/p2_scripter/Ve
rsión finalísima$ ./mygrep fich secreto extra
Uso: ./mygrep <ruta_fichero> <cadena_busqueda>
```

3. Conclusión

La realización de esta práctica nos ha sido útil para entender la sincronización de hijos, evitar hijos zombis, utilización de pipes... Gracias a la implementación a mano de un scripter podemos ver más a detalle cómo se manejan los comandos en linux y en general en los sistemas operativos.

Aunque nos haya costado bastante planificarlo, hemos conseguido realizar la parte del código capaz de procesar los n comandos unidos por pipes. Nos ha resultado difícil hacer el main para leer las líneas por como trata C a los buffers pero esto nos ha ayudado a comprender cómo los maneja la máquina. También hay que recalcar la dificultad que encontramos al intentar añadir la compilación del mygrep al Makefile.

La máquina virtual Gernika nos ha hecho cambiar el main ya que utilizábamos strcat y por algún motivo no funcionaba bien al ejecutarse dentro del servidor. Si bien es cierto que la práctica ha sido un poco laboriosa en general estamos contentos con el trabajo realizado y los conocimientos adquiridos.