Práctica 1

MiniShell

Rubén Moreno Martín

Jaime Llanos Melchor

Índice

Descripción del código

En primer lugar se han incluido diversas librerías para el correcto funcionamiento del código de la Shell, como son la librería “parser.h” proporcionada por el profesor, que es necesaria para el tratamiento de los comandos. También tenemos la librería “signal.h” con la que trabajaremos para el tratamiento de señales. Además tenemos las librerías “sys/types.h y sys/wait.h” para trabajar con procesos, así como la librería “fcntl.h” para incluir permisos a los ficheros que se abran. Por último tenemos las habituales “stdio.h, unistd.h, stdlib.h y string.h”.

También hemos considerado oportuno definir el tipo de dato tPipe, idea que al principio no considerábamos necesaria, pero que tras trabajar con un array de Pipes en un método, consideramos que el código sería más legible, y que se podría trabajar con él de forma más intuitiva si definíamos dicho tipo de dato. Además se han definido dos variables globales para ayudarnos a implementar correctamente el método Jobs: una es un array de datos del tipo “tline”, que guardará aquellas instrucciones que se ejecuten en background; se ha considerado oportuno determinar un tamaño máximo de diez instrucciones en el array, aunque se podría hacer redimensionados para hacer que su tamaño se vaya ajustando dinámicamente a un posible tamaño mayor. La otra variable global es un entero inicializado a 0 que nos servirá para la inserción de elementos en el array antes dicho, para que no se produzca ningún desbordamiento.

-**Método main**

El método main en primer lugar lo que hace es desactivar las señales SIGQUIT y SIGINT como así indica uno de los hitos del enunciado de la práctica. Posteriormente escribimos el prompt y declaramos el “tline” y el buffer que contendrá la entrada por teclado que reciba la miniShell.

En caso de que la llamada en consola a nuestra miniShell contenga argumentos, saltará una salida de error que indique que no debe tener argumentos el programa. De no tenerlos entraremos en un bucle infinito (que solo podrá acabar con la señal SIGSTOP (Ctrl + Z)) que recogerá en el buffer declarado con anterioridad, la entrada por teclado.

Dicha entrada se convertirá en una variable de tipo “tline” mediante la función “tokenize(char* buffer)” para así poder tratarla como una línea de comandos. Posteriormente comprobamos el número de comandos que se han introducido en nuestra miniShell. En caso de ser de un solo comando iremos al método “cd(tline *mandatos)” o al “unComando(tline *mandatos)”; y en caso contrario, que sea de varios comandos, iremos al método “variosComandos(tline *mandatos)”.

-**Métodos de redirección**

Tenemos tres métodos de redirección, el de entrada, el de salida y el de error.

En primer lugar lo que hacemos en estos métodos es comprobar si el mandato con el que estamos trabajando tiene algún tipo de redirección, y es que el tipo de dato “tline” tiene tres atributos (redirect_input, redirect_output, redirect_error) que en caso de que se produzca el

tipo de redirección, nos devolverá un puntero a un string que contenga el nombre del fichero para la redirección, y en caso contrario apuntará a NULL.

Luego hallaremos el descriptor de fichero del string que nos devuelva el atributo “redirect_*” con la función “open(char *algo, FLAGS)”. Donde FLAGS recogerá los permisos que tendrá el fichero que abrimos con open.

Por último haremos un “dup2(int fd, x)” donde “x = 0 | 1 | 2” para indicar que tipo de redirección se realiza sobre el fichero descrito por fd (que es nuestro mandato).

-**Método unComando**

En este método se creará un proceso hijo, el cual, en primer lugar activará las señales SIGQUIT y SIGINT para poder utilizarlas. Luego llamaremos a los métodos de redireccionamiento por si nuestro mandato hiciera uso de algún tipo de redirección. Después procederemos a la ejecución del mandato con el método “execvp(char *filename, char **argv)”.

Y por último en el proceso padre y después del “wait(NULL)” por el que espera al proceso hijo, desactivaremos de nuevo las señales SIGQUIT y SIGINT.

-**Método cd**

Es un método que podríamos haber incluido dentro del método “unComando”, pero que se decidió tratarse de manera individual para favorecer la legibilidad de nuestro código. Además dadas las características del comando “cd” tiene sentido también tratarlo de manera independiente al resto de comandos del sistema.

Lo que haremos en primer lugar es comprobar si ha recibido un directorio el comando, en caso de no haberlo recibido cambiaremos con “chdir(getenv("HOME")” al directorio dado por $HOME. Con “getenv” buscamos la cadena de caracteres que se le pasa en la lista de variables de entorno.

En caso de recibir un directorio, se cambiará a dicho directorio con “chdir”.

-**Método variosComandos**

Aquí en primer lugar crearemos un array de tPipes y otro de pids, y creamos los tPipes auxiliares p1 y p2. Luego definiremos todos los tPipes del array como pipes con la función pipe(int[2] p).

Después de activar las señales SIGQUIT y SIGINT, vamos a proceder a la creación de tantos hijos como comandos tenga la instrucción, y se trabajará con cada hijo. Tendremos tres casos diferentes:

El primero es que el hijo corresponda al primer mandato. En cuyo caso tendrá la redirección de entrada y escribirá en su pipe correspondiente la ejecución de dicho mandato.

Otro caso es que nuestro mandato sea el último mandato del pipe. En esta situación tendrá las redirecciones de salida y de error, y lo que hará será leer de la salida del último pipe lo que contenga, y ejecutar su mandato.

Y por último tenemos el caso de que nuestro mandato no sea ni el primero ni el último. Aquí leerá la salida del pipe anterior, y escribirá en la entrada de su pipe la ejecución de su mandato. Además se realiza la comprobación de que la instrucción no se ejecute en background, cumpliendo así el hito que establece que se deben poder ejecutar instrucciones de dos mandatos o más en background.

Finalmente, y ya fuera del bucle de creación de procesos hijos, tendremos un "wait(NULL)" para que el padre espere por todos sus hijos, y desactivamos las señales SIGQUIT y SIGINT.

-<u>Métodos Jobs y fg</u>

Para realizar el método Jobs se ha considerado utilizar las dos variables globales antes mencionadas. Se va a recorrer el array de "tline" en background y se va a ir imprimiendo por pantalla lo que corresponda. Este es el comportamiento que hubiéramos deseado para el método, por desgracia funciona erráticamente y no hemos podido solucionarlo del todo.

El método fg es el único que no hemos podido implementar ni parcialmente.

Comentarios personales

En la elaboración de esta práctica nos hemos encontrado con numerosos problemas que fácilmente se han solventado con el material teórico proporcionado con la asignatura. Pero la parte de la práctica que ha supuesto un verdadero reto es la de las instrucciones con varios comandos. Sobre todo a la hora de plantearnos como íbamos a utilizar los pipes, para lo cual tuvimos que esquematizar en varias ocasiones y de diferentes formas como íbamos a hacerlo.

Otra de las partes más difíciles, y que no hemos conseguido implementar correctamente como hubiéramos deseado, es la correspondiente al uso del background, que si bien consideramos que funciona, no podemos comprobarlo con exactitud por el mal funcionamiento del método Jobs.