Universidad Carlos III
Grado en ingeniería informática
Curso Sistemas Operativos 2023-24
Práctica 2
Curso 2023-24

Fecha: **16/04/2024** - ENTREGA: **2**

GRUPO: **81**

Alumnos: **Mario Ramos Salsón (100495849), Miguel Fidalgo García (100495770) y Víctor Martínez de las Heras (100495829)**

ÍNDICE

1. DESCRIPCIÓN DEL CÓDIGO

1.1 MANDATOS Y REDIRECCIONES SIMPLES

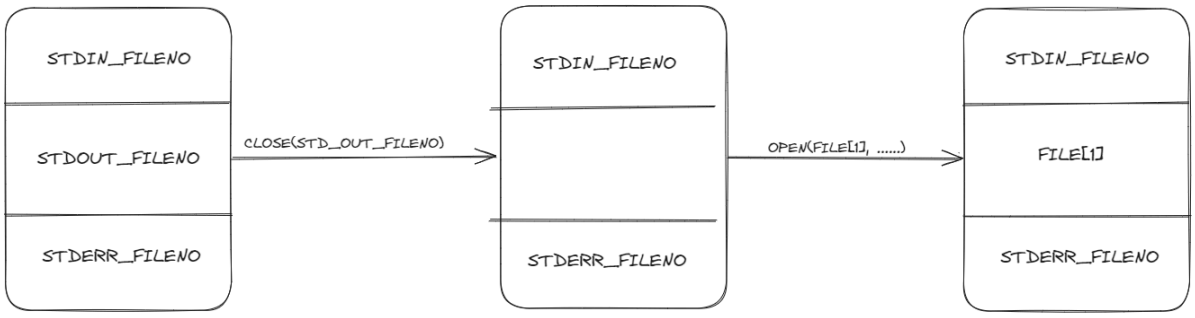
El programa msh.c, corresponde a una ***"minishell"*** programada por nosotros mismos. Para el correcto funcionamiento de esta, haremos uso de un "***parser"*** proporcionado por los profesores para agilizar cosas como la inserción en matrices de información o para ubicar donde se encuentran los mandatos a ejecutar con sus respectivos parámetros y ficheros de redirección.

Además, se nos proporcionan algunas funciones para agilizar la obtención de comandos de dichas matrices y para guardar dichos comandos en estructuras previamente definidas.

La organización de toda está práctica, ha girado en torno a la creación de pipes entre procesos hijo del mismo padre. La primera funcionalidad que hemos añadido en la práctica fue la ejecución por parte de un proceso hijo de un mandato simple con redirecciones a ficheros incluidas. Esto ha sido de gran ayuda ya que para posteriormente la inserción de mandatos anidados de forma infinita, todo su estructura iba a girar en torno a esto.

Para la ejecución de un mandato simple por parte de un hijo no crearemos ni un solo pipe ya que no es necesario. No obstante haremos uso de la función ***"fork()"***, para crear dicho hijo y de la función ***"execvp"*** para que ejecute el mandato pasado por parámetro. Este mandato se obtiene haciendo uso de la función ***"getCompleteCommand"***, a la cual le pasamos por parámetro la matriz "***argvv"***, y el número de comando que queremos obtener de ella, en este caso al ser un mandato simple le pasaremos el 0.

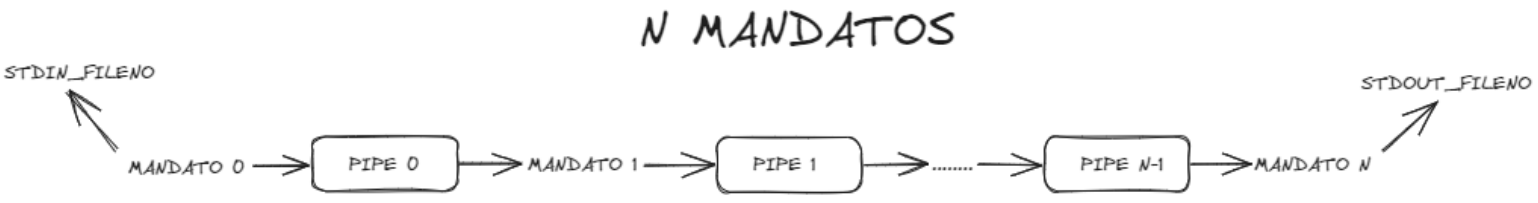
Para controlar la redirección, al ser solo un único hijo, simplemente comprobaremos si la matriz que almacena los ficheros de redirección está vacía en alguna de sus posiciones, y en caso de que no esté vacía cerraremos el descriptor de fichero correspondiente. En caso de se quiera redireccionar la entrada cerraremos con ***"close"*** el ***"STDIN_FILENO"*** y posteriormente se abrirá con ***"open"*** el fichero que queremos redireccionar para que ocupe su lugar en su descriptor de fichero. Este proceso se repetirá igual para la salida y la redirección de errores, ***"STDOUT_FILENO"*** y ***"STDERR_FILENO"*** respectivamente.



Para terminar con los mandatos simples, en el padre pondremos una condición para controlar si el mandato a ejecutar se tiene que ejecutar en ***"background"*** o no y así este hará ***"wait"*** o no. Cabe destacar que para todo este control de si estamos en el padre o el hijo, hacemos uso de un switch que cuando el ***"case"*** sea 0, estaremos en el hijo y cuando estemos en el ***"default"*** será el padre.

1.2 MANDATOS Y REDIRECCIONES COMPUESTAS

Para los mandatos compuestos, haremos uso de estructuras algo más complejas para su correcta ejecución. Lo primero de todo, creamos un array donde almacenamos todas las pipes que vamos a utilizar durante la ejecución del mandato. El número de pipes se calcula con el número de comandos a ejecutar menos uno, ya que se necesita una pipe menos.



Como se aprecia en el segundo diagrama, la salida del mandato ***"n"*** se enlaza con el pipe que se encuentra en el array en la posición ***"n"*** y su entrada con el pipe que se encuentra en la posición ***"n-1"***. Gracias a esto, conseguimos hacer

mandatos infinitamente anidados, ya que con el uso de un bucle ***"for"*** y analizando si estamos en el primer mandato, el último mandato, o cualquier otro mandato intermedio, podemos enlazarlos con sus correspondientes ***"pipes"***, para conseguir que los mandatos se comuniquen entre sí.

Como mencionamos anteriormente, el bucle ***"for"*** que controla todo esté fragmento de código, se divide internamente en 3 partes controladas por un ***"switch"***. La primera de todas, ***"case -1"*** se ejecuta en caso de que el ***"fork()"*** haya dado error. En la segunda ***"case 0"***, nos encontramos dentro del hijo y aquí es donde el bucle for entra en juego. En función de si estamos en la primera iteración es decir ***"i = 0"***, en la última ***"i = n-1"*** o cualquier otra, hacemos una cosa u otra. Para la primera de todas, enlazamos la salida del pipe con aquel que se encuentre en la posición 0 y su entrada será modificada en función de si hay una redirección de entrada o no al igual que se hacía con los mandatos simples. Para el último de todos, se enlaza su entrada con la del pipe que se encuentre en la posición ***"n-1"*** y su salida en función de si hay redirección. Para los mandatos intermedios, simplemente hacemos lo que se indica en el grafo superior. La salida se enlaza con el pipe correspondiente a su iteración, y la entrada con aquel que esté en la iteración -1.

Para la redirección de error, está se hace fuera de cualquier condición, ya que en el enunciado se indica que si existe este tipo de redirección, ha de afectar a todos ellos a la vez. Para poner fin al bloque del ***"switch"***, nos queda analizar al padre o ***"default"*** en el código. En este caso, el padre no hace nada, ya que no puede recoger a un hijo hasta que el resto no hayan acabado ya que están enlazados. Toda esta recolección se hace fuera del bucle cuando se acaba de ejecutar el mandato entero, y se hace igual que para los mandatos simples. Se analiza si hay que ejecutarlo en ***"background"*** o no, para realizar o no el ***"wait"***.

Queremos destacar como decisión de diseño la manera de recoger a los mandatos que se ejecutan en ***"background"***. La manera por la que hemos optado para recogerlos es cuando se ejecute uno en ***"foreground".*** Es decir cuando un proceso se ejecuta en ***"background"***, se mantendrá así hasta que se ejecute otro en ***"foreground"*** y así se aprovecha y se recoge a los dos a la vez.

1.3 MANDATOS INTERNOS

1.3.1 MYCALC

Para un correcto funcionamiento de este mandato, tenemos que analizar cada si el mandato que se ha introducido por la ***"minishell"*** empieza por ***"mycalc"***. Esto se debe a que al ser un mandato propio nuestro, no podemos pasarlo por la ejecución normal de cualquier otro mandato, ya que nos daría error. Para ello analizamos con un ***"if"*** si se ha introducido este mandato y activamos una variable llamada ***"internal"*** a 1 para evitar que entre en el resto de código de la práctica y se ejecute independientemente. Antes de realizar cualquier otra operación, analizamos si la estructura del mandato es correcta, ya que tiene que ser <mycalc><num1><add/mul/div><num2>. Para ellos recorremos las primeras cuatro posiciones de la matriz ***"argvv"***, y si ninguna de ellas en ***NULL***, en principio la estructura es correcta, aunque posteriormente haremos otro análisis. Una vez hecha esta comprobación, llamamos a la función auxiliar ***"mycalc"*** en la que gestionamos todas las operaciones. Para ello comprobamos si el segundo parámetro recibido se corresponde con una de las tres operaciones y si no devolveremos un error de formato.

Para la multiplicación y la división, simplemente operamos normal y mostramos el valor por la salida estándar de error como se indica en el enunciado, teniendo en cuenta de que no se puede dividir entre 0 y en caso de que se intente, lanzamos un error por la salida estándar.

Para la suma, hay que hacer unos ajustes previos, ya que hay que crear una variable de entorno ***"Acc"*** la cual ha de almacenar la suma de todas las sumas previamente hechas durante la ejecución. Esta se establece al principio del código mediante las funciones ***"sprintf"*** para pasar de ***int*** a ***char*** y ***"setenv"*** para declararla como variable de entorno. Esto hace que cada vez que se ejecute este mandato con una suma, haya que recoger dicha variable, pasarla a ***int*** con ***"atoi"***, operar y añadir a ***"Acc"*** la suma actual que se ha realizado. Posteriormente hay que volver a convertirla a texto y subirla como variable de entorno de nuevo con el valor actual insertado.

1.3.2 MYHISTORY

Para el mandato interno ***"myhistory"***, hemos hecho uso de dos funciones auxiliares para su correcto funcionamiento. La primera de ellas se encarga de imprimir los últimos "n" mandatos. En caso de que se hayan ejecutado más de 20, mostrará los últimos 20 y en caso de que no se hayan ejecutado tantos comandos, se ejecutará hasta el número actual de ellos. La segunda función se encarga de almacenar el mandato elegido en las matrices ***"argvv"*** y ***"filev"*** para que después se pueda ejecutar el mandato elegido.

Para guardar estos mandatos y que después puedan ser impresos o ejecutados de nuevo, hacemos uso de la matriz ***"history"*** proporcionada por los profesores. Esta, almacena como máximo 20 mandatos y en caso de que se ejecuten más mandatos, movemos manualmente todos los mandatos una posición a la izquierda para dejar la última posición libre para el mandato que se acaba de ejecutar.

No obstante, antes de llamar a estas dos funciones para que ejecuten su trabajo, comprobamos si el mandato que se acaba de introducir es un ***"myhistory"*** o no. Lo primero que hacemos es saber si se ha ejecutado un ***"myhistory"*** normal o se ha introducido un parámetro. En función de lo que haya ocurrido se ejecutará una función u otra.

Para la obtención de la información de las matrices para su posterior impresión o ejecución, hacemos un bucle. En caso de que se quiera imprimir los últimos mandatos ejecutados, hacemos un bucle que itera con una variable hasta que esta llegue al número de comandos ejecutados previamente. En caso de que se quiera volver a ejecutar un mandato, se hará el mismo bucle que terminará una vez que se hayan obtenido todos los comandos que se ejecutaron en dicho momento. Y para finalizar establecemos la variable ***"run_history"*** a 1 para evitar que a la siguiente ejecución ejecute el parser y remplace nuestra información.

2. BATERÍA DE PRUEBAS

Para la descripción de las pruebas usaremos el siguiente formato:

COMANDO EJECUTADO	DESCRIPCIÓN DE PRUEBA
RESULTADO OBTENIDO	

1. ls	El comando ls muestra una lista de los ficheros del directorio actual.
autores.txt libparser.so Makefile msh msh.c msh.o probador_ssoo_p2.sh ssoo_p2_100495849_100495829_100495770.zip	

2. ls > output.txt	Probamos la redirección de salida del mandato ls.
Se crea archivo output.txt con la salida del comando ls. Este fichero contiene la información de la prueba1. output.txt: autores.txt libparser.so Makefile msh msh.c msh.o output.txt probador_ssoo_p2.sh ssoo_p2_100495849_100495829_100495770.zip	

3. wc < autores.txt	Probamos la redirección de entrada del mandato wc. Se cuenta el número de palabras que contiene el archivo autores.txt
autores.txt: 100495849, Ramos Salsón, Mario 100495829, Martinez de las Heras, Victor 100495770, Fidalgo García, Miguel salida: MSH>>wc < autores.txt 2 14 107	

4. wc < autores.txt > output.txt	Probamos la redirección de entrada y salida con el mandato wc.
autores.txt: 100495849, Ramos Salsón, Mario 100495829, Martinez de las Heras, Victor 100495770, Fidalgo García, Miguel output.txt: 2 14 107	

5. wc < autores.txt > output.txt !> errores.txt	Probamos la redirección de entrada, salida y error simultáneamente sin que haya ningún error en el mandato.
autores.txt: 100495849, Ramos Salsón, Mario 100495829, Martinez de las Heras, Victor 100495770, Fidalgo García, Miguel output.txt: 2 14 107 error.txt: (VACIO)	

6. wc < noexiste.txt > output.txt !> errores.txt	Probamos la redirección de entrada, salida y error simultáneamente con un error en el mandato.
autores.txt: 100495849, Ramos Salsón, Mario 100495829, Martinez de las Heras, Victor 100495770, Fidalgo García, Miguel output.txt: (VACIO) error.txt: error: No such file or directory wc: 'entrada estándar': Descriptor de archivo erróneo wc: error de escritura: Descriptor de archivo erróneo	

7. ls | wc	Probamos la concatenación de dos mandatos. Ejemplo básico:
MSH>>ls | wc 8 8 112	

8. ls | wc > output.txt	Probamos la concatenación de dos mandatos pero con redirección de salida:
output.txt: 9 9 123	

9. ls | wc > output.txt !> errores.txt	Probamos la concatenación de dos mandatos pero con redirección de salida y de error, sin haber error en el mandato:
output.txt: 10 10 135	

10. ls | wccc > output.txt !> errores.txt	Probamos la concatenación de dos mandatos pero con redirección de salida y de error, con error en el mandato:
output.txt: (VACÍO) errores.txt: error: No such file or directory	

11. ls | grep u | wc	Probamos la concatenación de tres mandatos:
MSH>>ls | grep u | wc 2 2 23	

12. ls | grep u | wc > output.txt	Probamos la concatenación de tres mandatos con redirección de salida:
output.txt: 2 2 23	

13. ls | graaap u | wc > output.txt !> errores.txt	Probamos la concatenación de tres mandatos con redirección de salida y error con error en el mandato:
output.txt: 0 0 0 errores.txt: error: No such file or directory	

14. mycalc 3 add 2	Probamos la suma del mycalc
MSH>>mycalc 3 add 2 [OK] 3 + 2 = 5; Acc 5	

15. mycalc 3 mul 2	Probamos la multiplicación del mycalc
MSH>>mycalc 3 mul 2 [OK] 3 * 2 = 6	

16. mycalc 3 div 2	Probamos la división del mycalc
MSH>>mycalc 3 div 2 [OK] 3 / 2 = 1; Resto 1	

17. mycalc 3 div 0	Probamos la división entre 0 del mycalc
MSH>>mycalc 3 div 0 [ERROR] La división entre 0 no está permitida	

18. mycalc	Probamos la suma del mycalc con error de estructura
MSH>>mycalc [ERROR] La estructura del comando es mycalc<operando_1><add/mul/div><operando_2>	

19. mycalc 1 add	Probamos la suma del mycalc con error de estructura
MSH>>mycalc 1 add [ERROR] La estructura del comando es mycalc<operando_1><add/mul/div><operando_2>	

20. mycalc add 1	Probamos la suma del mycalc con error de estructura
MSH>>mycalc add 1 [ERROR] La estructura del comando es mycalc<operando_1><add/mul/div><operando_2>	

21. mycalc add 1 1	Probamos la suma del mycalc con error de estructura
MSH>>mycalc add 1 1 [ERROR] La estructura del comando es mycalc<operando_1><add/mul/div><operando_2>	

22. mycalc 1 add 2	Probamos la suma del mycalc
MSH>>mycalc 1 add 2 [OK] 1 + 2 = 3; Acc 3	

23. mycalc 1 add 10	Probamos la suma del mycalc con acarreo
MSH>>mycalc 1 add 10 [OK] 1 + 10 = 11; Acc 14	

24. mycalc 1 add -3	Probamos la suma del mycalc con número negativo y con acarreo
MSH>>mycalc 1 add -3 [OK] 1 + -3 = -2; Acc 12	

25. mycalc 1 ad 1	Probamos la suma del mycalc con error de estructura
MSH>>mycalc 1 ad 1 [ERROR] La estructura del comando es mycalc<operando_1><add/mul/div><operando_2>	

26. myhistory	Probamos la función myhistory
MSH>>myhistory 0 mycalc 3 div 2 1 mycalc 3 div 2 2 mycalc 3 div 0 3 mycalc 4 mycalc add 1 5 mycalc add 1 6 mycalc add 1 1 7 mycalc 1 add 2 8 mycalc 1 add 10 9 mycalc 1 add -3 10 mycalc 1 ad 1	

27. myhistory 0 (mycalc)	Ejecutamos uno de los mandatos ejecutados anteriormente a través de la función myhistory
MSH>>mycalc 3 div 2 [OK] 3 / 2 = 1; Resto 1 MSH>>myhistory 0 Ejecutando el comando 0 [OK] 3 / 2 = 1; Resto 1	

28. myhistory 0 wc < autores.txt	La estructura del mandato myhistory es incorrecta
MSH>>myhistory 0 wc < autores.txt La estructura del mandato myhistory es: myhistory / myhistory<valor>	

3. CONCLUSIONES

En cuanto a las conclusiones de la práctica, pensamos que ha sido una práctica relativamente sencilla que no ha consumido mucho tiempo. No obstante sí queremos destacar que los primeros días de su desarrollo, fueron lentos ya que no entendíamos bien que había que hacer exactamente y cómo funcionaban las cosas, pero una vez que lo hicimos, todo fue en buena dirección. No nos hemos encontrado con ningún problema a lo largo de todo el desarrollo de esta hasta casi el final con el mandato ***"myhistory"*** con parámetro.

Este ha sido el mayor problema que nos hemos encontrado y que no hemos sabido solucionar. Le hemos dedicado más de 20 horas y no han servido para nada. Pero al menos llegamos a la conclusión de que era algo relacionado con los punteros a memoria entre el parser y nuestro código. Hemos llegado a está conclusión tras un largo proceso de análisis en todas las partes del código, para observar cuando pasaba esto y cuando no. Por ello nos dimos cuenta de que justo antes de entrar al parser, las cosas estaban bien guardadas, pero una vez que se ejecutaba y terminaba, alguna que otra posición de la matriz ***"history"*** se veía alterada.

Tras este descubrimiento, decidimos no continuar el problema porque ya no se nos ocurrían más soluciones y nos veíamos incapaces de solucionar nada, ya que no teníamos acceso al parser para poder indagar más en él.

No obstante, queremos destacar la relativa sencillez de la práctica y el gran tiempo que se nos ha otorgado para realizarla ya que no hemos ido apurados en ningún momento. Además, consideramos que ha sido muy productiva y muy útil para entender plenamente cómo funciona el tema de las pipes, las llamadas al sistema y la redirección de las entradas y salidas tanto estándares como de error.