Práctica 2 Minishell

--Javier Bautista Rosell
-- NIA: 100315139
-- Email:100315139@alumnos.uc3m.es
--Oscar Senent Álvarez
--NIA: 100317422
--Email: 100317422@alumnos.uc3m.es

Índice

Contenido

Introducción

Se nos proponía la creación de una minishell, con la que pudiésemos ejecutar mandatos simples (ls, date, who, etc) y que a partir de ahí trabajásemos en ella para que implementase mandatos más complejos como pueden ser mandatos con redirecciones de fichero o mandatos simples agrupados en tuberías. La minishell, por otra parte, también tenía que ser capaz de ejecutar mandatos en background si así lo especificábamos con un "&", y por ultimo también tenía que ser capaz de ejecutar un último mandato que nos planteasteis, (myhistory), el cual tenía que mostrarnos en la terminal todos los últimos mandatos que habíamos introducido y ejecutado además de sus redirecciones de ficheros y si estos mandatos estaban ejecutándose o no en background.

Descripción del funcionamiento:

Nuestro programa lo hemos dividido en partes según el número de mandatos que introducimos de manera que diferenciamos entre la ejecución de un solo mandato, de dos mandatos que necesiten una tubería y de tres mandatos que necesiten dos tuberías.

El programa está basado en el parser que vosotros nos proporcionasteis, el cual nos permitía almacenar los mandatos y ficheros en dos vectores, de manera que pudiésemos acceder a ellos y ejecutarlos con la función execvp. Estos dos vectores son el argvv[][] que almacena cada uno de los mandatos estén o no estén en tuberías, y el filevv[], que nos almacenara, si existen, en la posición 0 el fichero de entrada del mandato, en la posición 1 el fichero de salida y en la posición 2 la redirección de error.

Planteamiento:

Solo 1 argumento:

Si se introduce un solo mandato entonces argvv[1]==NULL, y nos centraremos en trabajar con argvv[0]. Primero tendremos que crear un hijo (pid=fork()), ya que en todos los casos será el hijo el que ejecute el mandato y nunca el padre, ya que se daría por eliminado el proceso.
Para la ejecución del mandato en el hijo solo tendremos que llamar a un execvp que nos ejecute el contenido del vector. Además en el hijo, antes de ejecutar el mandato deberemos comprobar si existe algún tipo de redirección y para ello solo tenemos que ver si alguno de los espacios del filevv[] es diferente de NULL.
Si nos encontramos con que hay una redirección de entrada o salida solo tendremos que cerrar la entrada o salida estándar y después proceder a abrir el fichero en modo lectura, si es fichero entrante o en modo escritura si es saliente.
La salida de error funcionara de la misma manera, cerrando la salida de error estándar y almacenándose en el archivo, el mensaje de error que nos proporciona el terminal.
Por ultimo en el caso del padre tendremos que controlar si el proceso se encuentra ejecutándose en background o no. Si se está ejecutando en background entonces bg será diferente de 1 y tendremos que hacer al padre esperar con un wait hasta que termine el hijo. Si no se ejecuta en background tendremos que imprimir el pid del proceso hijo.

Si el número de mandatos introducidos asciende a dos necesitamos hacer uso de una tubería que nos

conecte ambos mandatos, de manera que el primer mandato genere la entrada del segundo mandato. Para conectar correctamente la tubería tendríamos que conectar en un primer hijo la salida estándar del mandato con la entrada de la tubería con la llamada dup o dup2, y cerrar todas las demás entradas y salidas, antes de ejecutar este primer mandato.

Pruebas:

http://gyazo.com/4bb9b8319936e5f7145be0116db438a5
http://gyazo.com/a903b3e88f4b4afb4b07b0af61aa13fd
http://gyazo.com/482e3f20223c7905f54b1be9c1b385b8
http://gyazo.com/642eb88b431669645cf9a89af3343525
http://gyazo.com/f3d8d089a1b62b492ae93f9832d88a80
http://gyazo.com/74c71ea4518f486698fbb3f31ac255dd

Con 2 argumentos:

Para ejecutar el segundo mandato, antes tenemos que crear un segundo hijo, en el que conectamos la entrada estándar del mandato con la salida de la tubería creada con anterioridad, pasando así la información del primer mandato al segundo mandato. Ya solo nos quedaría ejecutar este segundo mandato con otro execvp y comprobar si la ejecución es en background o no. También aquí tendremos que añadir la redirección de ficheros de la misma forma que en el ejercicio anterior, conectando la entrada estándar del primer fichero o la salida estándar del segundo con el fichero seleccionado.

Pruebas:

http://gyazo.com/929087401cffe5c0c6bf7cd2ff233552
http://gyazo.com/45236fbec03d17697e3a2f5787c212bf

Con 3 argumentos:

Para ejecutar una secuencia de tres mandatos tendríamos que crear una nueva tubería que conectase el segundo mandato con el tercero, reciclando por lo demás el código de la secuencia de dos mandatos. Para ello duplicamos la salida estándar del segundo mandato con la entrada de la segunda tubería y la salida de la tubería con la entrada estándar del tercer mandato. Esto último junto con la ejecución del tercer mandato pasaríamos a colocarlo en un tercer hijo. Terminaríamos esta fase con la introducción de las redirecciones y de los background.

Comenzamos inicializando un string de gran tamaño al que llamamos “buffer” en el que vamos a ir guardando mediante un for junto con un strcpy que nos permite el desplazamiento en argvv mientras vamos imprimiendo cada uno en el string para finalmente hacer un write y guardar todos estos mandatos del buffer en un file al que después nombraremos como historial, y que es el mismo que imprimiremos con un “cat” si el usuario introduce el mandato myhistory.
Ademas tenemos que crear strings para cada uno de los símbolos que necesitemos para expresar los mandatos. Cuando ya estén iniciados solo tenemos que empezar a recorrer if para ver si hay alguna redirección de ficheros (filevv[0] o filevv[1] diferente de null) o si se está ejecutando el mandato en background.
Si hay redirecciones entonces solo tendremos que hacer un strcpy, en “historial”, copiando primero el símbolo de la redirección correspondiente y después copiar el fichero correspondiente a la redirección.

Pruebas:

http://gyazo.com/9e86297b22b1c0a64948b7465190758b
http://gyazo.com/3e954404d3405758846cd9bd0c3fbad2

MyHistory:

Por ultimo en el myhistory también tenemos que incluir los mandatos que contengan tuberías y para ello usamos un vector más que tenga en su interior el símbolo "|", para que según vayamos imprimiendo los mandatos de la tubería, se vayan separando con | imitando así el funcionamiento de myhistory.

Pruebas:

http://gyazo.com/b47bab5e17830358f8a47bc10f399291

Conclusión

Tuvimos algunos problemas al principio ya que costo algo entender el funcionamiento delos vectores creados por el parser que nos proporcionasteis pero esto fue fácilmente resuelto.
Los problemas más importantes se nos aparecieron cuando comenzamos a programar las tuberías ya que por alguna razón no nos funcionaban los programas devolviéndonos un mensaje de "violación de segmento". Esto nos costó un poco más comprobarlo y corregirlo ya que fue difícil encontrar donde podíamos estar cometiendo el error, dándonos cuenta finalmente que faltaban varios close en los pipes y esto afectaba al funcionamiento.

Otro error relacionado con las tuberías fue que cuando ejecutábamos un mandato con tuberías en el msh, se nos cerraba el msh al terminar de ejecutar el mandato. Este error se debía a que como el programa estaba en fase de pruebas habíamos metido algún execvp en los padres y esto hacia que el proceso finalizara, saliendo así del msh. La resolución fue mucho más fácil ya que solo tuvimos que cambiar nuestros execvp para que siempre ejecutáramos únicamente en los hijos y nunca en los padres.