24-4-2016

Minishell

Sistemas Operativos

Susana Pineda De Luelmo

Índice:

1. Autores:

Susana Pineda De Luelmo

2. Descripción del código, Objetivos parciales.

- **Prompt y bucle.**

Lo primero que tenemos que hacer es un bucle que se repita cada vez que tengamos una entrada por teclado. Para ello, creamos un bucle while que se repetirá cada vez que obtengamos una entrada de teclado y guardamos esa información en un buffer.
La información almacenada en el buffer (lo que obtenemos por teclado) lo vamos a transformar mediante la librería incluida en la práctica para facilitar las operaciones a realizar.
Dentro del while tendremos las diferentes posibilidades que pueden ocurrir cuando leemos un comando de teclado:

- La línea no contiene nada, entonces mostramos otra vez el prompt y esperamos por la siguiente instrucción.
- La línea tiene un único mandato. En este caso pasamos al método ejecutar1, el cual se encargará de la ejecución del mandato. Para mayor claridad del código se han separado las diferentes ejecuciones en métodos y desde el main lo único que hacemos es llamarlos y pasarles la estructura tline en la que almacenamos toda la información de la línea introducida por teclado. Más adelante se le añadirá la comprobación de que el mandato sea o no sea cd.
- La línea tiene dos mandatos.
- La línea tiene más de dos mandatos.

Para mayor comodidad se han separado las diferentes ejecuciones según el número de mandatos que tengan, pero después de implementar el método nmandatos(line) se podrían eliminar los otros dos y realizarlos desde el mismo método.
Por ultimo tendríamos que mostrar el prompt, el cual en un principio simplemente era un símbolo (-->) pero finalmente para comprobar el funcionamiento del mandato cd pasó a ser la ruta absoluta del directorio actual seguido de &. Para ello, lo único que tenemos que hacer es obtener el directorio actual mediante `getcwd(cwd, sizeof(cwd))` seguido del signo & y un espacio y mostrarlo por pantalla antes del bucle while para la primera ejecución y al final del mismo para el resto de ejecuciones.
De esta forma quedaría implementada la estructura general de nuestra Shell, a partir de ahora lo que tendremos que hacer es implementar los métodos que se encargan de ejecutar 1 mandato (ejecutar1(line)), de ejecutar 2 mandatos (ejecutar2(line)), de ejecutar más de dos (ejecutarn(line)) (estos tres resumibles en ejecutarn una vez implementado), ejecutar el comando cd y la gestión de las redirecciones de entrada y de salida.

- **Reconocer y ejecutar en foreground líneas con un solo mandato y 0 o más argumentos.**

Para este apartado creamos el método ejecutar1 que recibe como argumento un estructura tline en la cual tendremos toda la información necesaria para ejecutar nuestro mandato.
Para ejecutar un solo mandato creamos un hijo mediante fork(), el cual se encargará de ejecutar el mandato que hemos pasado line. Comprobamos que no ha habido ningún error al crear el hijo, en caso contrario mostramos un error por pantalla y salimos de la ejecución. Si el hijo se ha creado

correctamente cuando lo estemos ejecutando (pid == 0) ejecutaremos el mandato mediante la siguiente instrucción:

```
execvp(line->commands[0].filename, line->commands[0].argv)
```

la función execvp recibe como argumentos el nombre del mandato a ejecutar y un puntero a los argumentos del mandato. Los cuales se los pasamos mediante la estructura que nos proporciona la librería incluida en la práctica.

A continuación de la ejecución mostramos un mensaje de error, ya que si ese error se muestra es que la ejecución ha terminado de forma incorrecta.

Mientras tanto, el proceso padre espera a que el hijo termine y nos indique como ha terminado. En caso de que este termine mal se muestra un mensaje por pantalla.

- **Redirección de entrada y de salida de un solo mandato**

En mi caso he separado en dos métodos diferentes las redirecciones de entrada y de salida para facilitar la implementación de las redirecciones en ejecuciones de dos o más mandatos.

  - gestionRedireccionEntrada(tline *line):

    La función recibe como argumento una estructura tline en la cual tendrá toda la información necesaria.

    Lo primero que tenemos que hacer, después de comprobar si hay redirección de entrada, es abrir el archivo al que queremos redireccionar la entrada, en este caso lo abriremos solo como lectura y almacenaremos su descriptor de fichero en una variable de tipo int (fdE). Comprobamos que fdE es distinto de -1, es decir, no hemos tenido ningún error al abrir el archivo, en caso contrario mostramos un mensaje de error por pantalla. Si el archivo se ha abierto correctamente lo siguiente que tenemos que hacer es redireccionar la entrada estar al descriptor de fichero mediante la función dup2(), de esta forma la entrada estándar a partir de este momento, dentro de este proceso, será el fichero con el descriptor de fichero fdE. Una vez redireccionada la entrada cerramos el archivo y comprobamos que este se haya cerrado correctamente. Por ultimo comprobaremos que no ha habido ningún error durante la gestión de la redirección de entrada.

  - gestionRedireccionSalida(tline *line):

    La función será muy parecida a la redirección de entrada, pero en este caso, después de comprobar que hay redirección de salida, abriremos el archivo en el que vamos a escribir solo como escritura y en caso de que este no exista se crea uno nuevo con el nombre indicado y con permisos. El descriptor de fichero de este nuevo archivo lo guardaremos una variable de tipo int (fdS). Una vez abierto o creado el archivo comprobaremos que no ha habido ningún error durante este proceso. Si ha habido algún problema se notificara mediante un mensaje por pantalla y en caso contrario redireccionaremos la salida estándar al archivo cuyo descriptor de fichero sea fdS mediante la función dup2(). Una vez redireccionado cerramos el fichero en el que hemos escrito y comprobamos que se ha cerrado correctamente.

Una vez creados los métodos encargados de la gestión de las redirecciones de entrada y de salida lo único que tenemos que hacer es llamarlos, para comprobar si hay redirección de entrada o salida, antes de ejecutar nuestro mandato. De esta forma si hay redirecciones estas se mantendrán hasta que se ejecute el mandato que queremos.

- **Reconocer y ejecutar en foreground líneas con dos mandatos, enlazados con '|' y con posibles redirecciones de entrada o de salida.**

Para ejecutar dos mandatos necesitamos dos procesos hijos que se comuniquen mediante un pipe, uno de los hijos se encargará de escribir en el pipe, y el otro se encargará de leer del mismo.
Lo primero que tenemos que hacer es declarar los dos pid_t de los dos hijos y la tubería p que los va a comunicar.

  - Hijo 1:

    Creamos el primer hijo que va a ser el encargado de escribir en el pipe. Como este hijo (pid1) no va a leer del pipe cerramos la salida de este ya que no la vamos a utilizar. A continuación, redireccionamos la salida estándar a la entrada del pipe para que se escriba en este mediante la función dup2() al igual que en las redirecciones a fichero.
    Una vez que hemos cerrado las partes del pipe que no vamos a utilizar y hemos redireccionado la salida del primer hijo al pipe tenemos que comprobar si hay redirección de entrada (solo puede haber redirección de entrada en el primer hijo) para comprobarlo lo único que tenemos que hacer es llamar al método gestionRedireccionEntrada(line). Por ultimo ejecutamos el mandato, al ser el mandato que escribe en el pipe, es el primero que aparece en el array de comandos que nos proporciona la librería (line->commands[0])
  - Hijo 2:

    A continuación pasamos a crear el hijo 2 (pid2) que será el encargado de leer del pipe. Como se va a encargar de leer del pipe, tenemos que cerrar la entrada del pipe y tenemos que redireccionar la entrada estándar a la salida del pipe. Una vez que tenemos hecho esto comprobamos si hay redirecciones de salida, solo se pueden dar en el último mandato de la línea, llamando al método gestionRedireccionSalida(line). Por ultimo ejecutamos el mandato, en este caso será el segundo mandato que nos aparezca en el array comandos (line->commands[1]).

Mientras se ejecutan los hijos, el padre esperara por ellos y una vez terminados cerrará el pipe.
Como no sabemos que hijo se va a ejecutar primero las tuberías nos sirven para sincronizar los procesos, hasta que el hijo1 no se ejecute y escriba algo en el pipe el hijo 2 no podrá ejecutarse.

De esta forma quedaría implementada la ejecución de dos mandatos con redirección de entrada y de salida.

- **Reconocer y ejecutar líneas con más de dos mandatos y con posibles redirecciones de entrada o salida.**

Esta parte de la práctica fue la más complicada y acabé haciendo más de una implementación para esta parte, aunque finalmente solo funciona una, posiblemente por problemas al abrir y cerrar los pipes voy a explicar algunas de las implementaciones:

  - Intento 1:

    La primera idea fue crear un array de hijos (cada uno de ellos se encargaba de ejecutar un mandato) como solo necesitábamos pasarnos la información de un proceso a otro esto se podía hacer con un solo pipe, el cual recibiese la información de hijo[i] y le pasase la información a hijo [i+1]. De esta forma cuando estamos ejecutando hijo[i] este se encarga de escribir en el pipe, es decir, cerramos la salida del pipe, redireccionamos la salida estándar a la entrada del pipe y ejecutamos, en el caso de que i sea 0, es decir sea el primer mandato comprobamos si hay redirección de entrada antes de ejecutar. Por otra parte el hijo [i+1] sería el encargado de leer del pipe y en el caso de no ser el último de escribir en el mismo. Por lo tanto, si no era último, no cerraba nada y redireccionaba la entrada estándar

a la salida del pipe y la salida estándar al pipe y ejecutaba, por el contrario, si hijo[i] era el último se cerraba la salida del pipe y se redireccionaba la entrada estándar a la salida del pipe.
Mientras tanto el padre esperaba a que los procesos terminasen. Este intento no funcionó porque se quedaba en un bucle y no terminaba nunca la ejecución. Por lo tanto pasé a otra implementación.

- Intento 2:

La siguiente implementación fue con un array de hijos y un array de pipes, de forma que hubiese el mismo número de hijos que de comandos y un pipe menos que comandos. Lo primero que hacía era crear los pipes y después dentro de un bucle for ir creando y ejecutando los diferentes hijos. Si i = 0, es el primer comando, redirecciono la salida estándar al pipe con el mismo índice y compruebo si hay redirecciones de entrada, a continuación ejecuto y cierro todos los pipes. Si es uno de los comandos intermedios redirecciono la entrada estándar a la salida del pipe con índice i-1 y la salida estándar a la entrada del pipe con el mismo índice. Ejecuto y cierro todos los pipes. Por ultimo si es el último comando redirecciono la entrada a la salida del pipe anterior, ejecuto y cierro todos los pipes. El padre espera a los procesos y cierra todos los pipes. El intento tampoco funcionó seguramente por cerrar algo que no había que cerrar o por no cerrar algo que había que cerrar. Con esta estructura intenté varias formas de cerrar los pipes, cerrando primero lo que no utilizaba y luego lo que había utilizado y cerrando todo junto justo después de redireccionar, pero no funcionaba.

- Intento 3:

La última implementación fue con dos pipes y con un solo hijo. Creamos una tubería principal y después un bucle que se repite una vez por cada mandato que hay. Si estamos en un mandato intermedio creamos otro pipe que nos va a servir de intermediario para que no se nos quede en un bucle infinito como ocurría en el primer intento. Luego, creamos el hijo que nos va a servir para ejecutar el mandato, como lo creamos dentro del bucle no nos hacen falta más. Si se ha creado bien el hijo pasamos a comprobar en que posición nos encontramos. Si es el primer mandato comprobamos si hay redirección de entrada, cerramos la salida del pipe principal, redireccionamos la salida estándar a la entrada estándar del pipe y la cerramos, ya que no nos va a hacer falta y ejecutamos. Si nos encontramos en un mandato intermedio redireccionamos la entrada estándar a la salida del pipe y lo cerramos, ahora utilizamos el pipe auxiliar para almacenar la salida, cerramos la salida del pipe auxiliar y redireccionamos la salida estándar a la entrada del pipe, por ultimo cerramos el pipe y ejecutamos. Si nos encontramos en el último mandato redireccionamos la entrada estándar a la salida del pipe, comprobamos si hay redirecciones de salida y cerramos el pipe, por último ejecutamos el mandato.
El padre de cada ejecución se encarga de que si nos encontramos en el primer mandato se cierre el pipe p y si nos encontramos en los intermedios copiamos la información del pipe p en el pipe auxiliar para encadenar la información de una ejecución a la siguiente. Una vez que termina el for (todos los hijos) esperamos a que terminen todos, cerramos los pipes y esperamos a status que nos indique como ha terminado la ejecución.

A parte de estas implementaciones intenté también hacerlo de forma recursiva y con señales pero el seguimiento era mucho más complicado y tampoco funcionaba.

El intento 2 funcionaba pero solo con algunos mandatos, los menos exigentes, mientras que el intento 3 ya funciona con todos los mandatos, con argumentos y con redirecciones de entrada y de salida. Es el implementado en la práctica entregada.

- **Comando cd:**

Dentro del main contemplamos la posibilidad de que dentro de la ejecución de un solo mandato este pueda ser el mandato cd, el cual tenemos que implementar aparte. Por esta razón, en el main comprobamos si ese único mandato es cd, en caso afirmativo llamamos a la función encargada de la gestión de este comando (gestionCd). Dentro de gestionCd tenemos diferentes posibilidades, si solo tenemos cd tenemos que cambiar a HOME, si cd va seguido de " .. " nos dirigiremos al padre y si va seguido de una ruta nos dirigiremos a ella.

  - Solo cd:
    Si solo tenemos cd, es decir, no tenemos ningún argumento en cd, haremos un chdir a la variable home, para conseguir el valor de esta variable contamos con la función getenv que nos devuelve la ruta absoluta de la variable introducida.
  - Cd seguido de "..":
    Si cd esta seguido de " .. " tenemos que cambiar al padre del directorio en el que nos encontremos, para ello contamos con la función dirname que nos devuelve la dirección del padre de la variable que le pasemos como argumento. En nuestro caso, para que no nos de error, hemos creado una variable en el main que se pasa como argumento a la función que nos indica la dirección en la cual nos encontramos en la Shell, no donde se encuentra el archivo (problema que nos encontrábamos cuando utilizábamos getenv()).
  - Cd seguido de una ruta:
    Por ultimo si cd esta seguido por una ruta relativa o absoluta lo único que tiene que hacer es cambiar a dicha ruta.

De esta forma queda implementada la gestión del mandato cd.

- **Evitar que los comandos lanzados en background y la minishell mueran al enviar las señales desde teclado SIGINT y SIGQUIT, mientras que los procesos en foreground respondan ante ellas.**

Para evitar que los comandos lanzados en background y la minishell mueran al enviar las señales SIGINT y SIGQUIT lo único que tenemos que hacer es indicar en la minishell que se ignoren esas dos señales y en el identificador de background que haga lo mismo, por el contrario, en los procesos en foreground le indicamos que si recibe esas señales responda ante ellas.
Para indicar que tiene que ignorar esas señales solo necesitamos dos líneas de código:

```
signal(SIGINT,SIG_IGN);
signal(SIGQUIT,SIG_IGN);
```

Estas dos lineas las pegaremos en el main y en los procesos en background (en mi caso sin implementar)

Por el contrario, si queremos que estas señales no sean ignoradas pegaremos las dos líneas de código siguientes:

```
signal(SIGINT,SIG_DFL);
signal(SIGQUIT,SIG_DFL);
```

Estas dos lineas las pegaremos en los métodos ejecutar1(), ejecutar2() y ejecutarn(), de esta forma nos aseguramos de que estos procesos si hacen caso a las señales.

3. Comentarios personales:

La práctica en general me parece muy buena forma de asentar los conocimientos que teníamos de las clases y de los ejercicios. Pero la parte de más de dos mandatos me parece que conlleva demasiado tiempo y su mayor complicación es cerrar los pipes bien, lo cual también se puede hacer con otros ejercicios que no consumen tanto tiempo en fallos pequeños o tan difíciles de ver.

Personalmente, la práctica me ha llevado más tiempo del esperado por la parte de más de dos mandatos y no he podido implementarla entera, pero por lo demás creo que me ha venido muy bien para asentar conocimientos.

Por otra parte, me parece que podría venir bien en la descripción de la práctica otra especie de apéndice que avise de que hay algunos mandatos que son más difíciles de ejecutar o que pueden dar fallo con más facilidad para que al comprobar el funcionamiento de la práctica podamos utilizar esos mandatos directamente.