Escuela Técnica Superior de Ingeniería Informática

Asignatura: Sistemas Operativos

Práctica 1 : Minishell

Martínez-Aedo Gómez, Juan
Valor Cano, Alejandro

Índice

1. Introducción

Planteamos la práctica como una forma de ver nuestros conocimientos y una forma de trabajar ya en los conocimientos que nos tendremos que examinar en enero.
Decidimos usar la extensión de Visual Studio Code, Live Share de trabajo colaborativo, pudiendo trabajar de una forma cómoda, en remoto y bastante dinámica. Por lo que, además de aprender sobre temario de la asignatura, pudimos conocer nuevas herramientas de trabajo, para el futuro.

2. Descripción del código.

A partir del archivo test.c dado, creamos nuestro main usándolo de forma de controlador de las entradas. Nuestro main realiza un bucle sin tope de repeticiones, esperando la entrada. Se lee la entrada, se pasa por el parser dado y se tokeniza como hace normalmente la Shell. Esto nos da una variable tipo tline que nos ayuda a controlar errores, redirecciones, ejecuciones en segundo plano y comandos externos e internos. En primera instancia se comprueban las redirecciones de entrada, salida y error. En caso de que haya redirección en el tline lanzamos las función/es correspondiente. Vuelve al main y se comprueba el mandato o mandatos que se han lanzado. En el propio main comprobamos que sean mandatos reconocidos (o bien mandatos externos de la Shell, o bien los internos pedidos desarrollar: cd , jobs ,fg), pero no sus argumentos, ya que esto en la función encargada de la ejecución del mandato con un argumento erróneo, nos comunica el posible error. Para finalizar, si existe una redirección reestablecemos la inicial de nuestro main, para que la redirección se produzca exclusivamente en el mandato especificado .Por último, escribimos el directorio actual, actualizándose si se ha ejecutado el mandato cd con el prompt propio de esta Minishell : “msh>”.

- **Redireccion_entrada, redireccion_salida y redireccion_error:**
  Abrimos un descriptor de ficheros con la redirección dada y con los permisos necesarios en cada caso. Controlamos posibles errores y, si no hay, reescribimos el descriptor. Al volver al main y antes de terminar el bucle, si ha habido redirecciones, las reestablecemos a las estándar. De esta forma nos aseguramos de que la redirección sea exclusiva del mandato ejecutado con redirección.

- **Redireccion_1comando:**
  En este caso lo diferenciamos de varios comandos debido a que no usamos pipes. Creamos el proceso hijo y comprobamos que no ha habido error en la creación. En el código que ejecuta el hijo lo redireccionamos a redireccion_bg(se explica en detalle más adelante), que comprueba el modo de ejecución del mandato y le asigna las respuesta correspondiente a las señales especificadas en el enunciado. Ejecutamos el comando dado con el execvp(), que se encarga de devolvernos los posibles errores en la ejecución, y si va bien, nos devuelve la ejecución del mandato por pantalla (dado que es la salida estándar del padre).

- **Redireccion_varios_comandos:**
  A diferencia de la función de un comando, este se caracteriza por el uso de pipes. Para facilitarnos el control, y evitar errores de C, hemos decidido crear todas las pipes que vamos a usar, antes que los hijos. Esto a su vez nos ha supuesto ejecutar un bucle para

cerrar las pipes que no usa cada hijo, ejecutándose cada vez que se ejecuta el código del hijo. Hemos hecho una diferencia entre el primer hijo, posibles hijos intermedios e hijo final, todo ello debido al patrón de pipes que usa cada hijo. El primer hijo sólo escribe en el primer pipe, los hijos intermedios leen del pipe previo a él y escriben en el siguiente, y el hijo final sólo lee del último pipe creado. Como ha heredado los descriptores del padre, la salida es la misma que el padre en ese mismo momento (esto implica que, si existe redirección, el padre ya ha cambiado sus descriptores y, por tanto, el hijo los ha heredado). Todos ellos realizan la ejecución de su mandato, además de comprobar redirecciones de salida.

- **Redireccion_bg:**
  Comprueba si el campo en tline de background es 1, en ese caso, las señales del proceso desde el que es llamada pasan a ser ignoradas. En caso de no ser 1, las señales SIGINT y SIGQUIT responden a ellas por defecto.

- **Redireccion_fg y Redireccion_jobs:**
  Estos no han sido implementados, dado que se nos ha atragantado y no hemos conseguido que nos funcionara. En primer lugar, al ejecutar en background cualquier mandato, añadiríamos ese proceso a una lista de datos tipo tline. Al ejecutar Redireccion_jobs, se mostraría por la salida estándar la lista de procesos en background , su estado, y el/los mandatos del proceso. Estos, estarían numerados, y con el comando Redireccion_fg con el número del job al que nos refiriéramos, cambiaríamos las señales de ese mandato, y lo eliminaríamos de la lista de datos tline.

3. Valoración Personal

En aspectos generales, la práctica ha servido para afianzar muchos conceptos de los primeros temas que se nos habían podido quedar un poco a medias (p.e., las diferencias exactas entre comandos externos e internos). En sí, la práctica no es demasiado larga y se focaliza claramente en el objetivo de esta parte de la asignatura (dándonos el parser, evitando tener que controlar strings), entender bien la programación en C, los descriptores de ficheros, creaciones de procesos y las comunicaciones entre ellos. La práctica es densa, y las dificultadas de hacer debug con C, y en especial, en la ejecución de los hijos, ha resultado bastante tedioso encontrar fallos. Los ejercicios que se han ido haciendo en clase han resultado bastante de utilidad, ya que se han centrado en aspectos bastantes concretos, siendo muy útiles a la hora de plantear pequeños controles de errores o el uso de funciones ya incluidas en librerías, que hemos usado.

Por otro lado, hemos echado en falta un poco más de información en detalle (ya sean en el enunciado o en la teoría), sobre la diferencia entre comandos ejecutados en background y foreground, más allá de cómo responden a las señales SIGQUIT y SIGINIT. Así como los ejercicios propuestos en clase han sido de mucha utilidad, también hemos echado en falta mejores conocimientos en el uso de memoria dinámica en C, dado que es necesario en los mandatos fg y jobs para controlar una lista de mandatos (ya convertidos en procesos) en ejecución en 2º plano.