Práctica Obligatoria 2
Programación en C

Sistemas Operativos 2017/18 – Grado en Ingeniería del Software

Iván Pérez Huete – Carlos Olmo Sahuquillo
12-7-2017

Tabla de contenido

1. Introducción

La práctica consiste en la implementación de un intérprete de mandatos (consola o minishell) en el lenguaje de programación C que pueda entender y ejecutar comandos leyéndolos a través del teclado.

Además, había que capacitar a la minishell para:

- Poder ejecutar uno o varios mandatos separados entre el carácter "|".
- Tolerar redirecciones tales como:
    a. De entrada (que se pasaría al final del comando con un < fichero).
    b. De salida (que se pasaría al final del comando con un > fichero).
    c. De error (que se pasaría al final del comando con un >& fichero).
- Tener la posibilidad de mandar comandos a ejecutarse en background.

Posteriormente describiremos más detalladamente lo que debíamos hacer con la minishell y un comentario con la explicación de la implementación y de los problemas encontrados en la realización de la práctica.

Junto a este documento, que corresponde a la memoria de la segunda practica obligatoria de la asignatura de Sistemas Operativos, se adjuntará los siguientes archivos:

- Libparser_64.a, librería proporcionada de antemano.
- myshell.c, la propia implementación de la minishell.
- parser.h , proporcionado de antemano, necesario para compilar la minishell.

2. Implementación de minishell.c

2.1 Reconocer y ejecutar líneas con un solo mandato

Ejemplo del comando: **ls - l**

2.1.1 Implementación

Para implementar este objetivo, nuestra minishell trata de mirar si el comando que le pasas es válido, en caso de no devolver un NULL, ejecutaremos el comando execvp que mirará la línea de comandos que le hemos pasado y tratará de pasar ese comando como argumento. En el caso de que Linux tenga ese comando devolverá un 1 y saldrá de la función.

2.1.2 Comentarios adicionales

En este apartado tuvimos poco problema, ya que tomamos como referencia el ejercicio de "ejecuta" que habíamos hecho previamente en el tema 4. En su momento aprendimos como trabajar con execvp que es lo principal para este apartado y este conocimiento nos beneficiará para próximos apartados.

2.2 Redireccionar desde entrada estándar y a salida estándar

Ejemplo del comando: **ls > archivo**

2.2.1 Implementación

Nuestro programa mirará si la entrada es distinta a NULL en caso de ser así, usaremos el comando open para abrir el archivo que nos llegó. Si el comando open nos devuelve un -1 habrá fallado y por lo tanto habrá que comunicarlo al cliente con un stderr. En caso de que no haya fallado, usaremos el comando de dup2 para llevarlo a la entrada. En el caso de que estuviéramos pidiendo que lo redirigiera a una salida determinada (a un archivo), sería lo mismo, pero esta vez cambiando el método open, por un creat, cada uno con sus permisos necesarios para que se pueda escribir o leer en ellos.

2.2.2 Comentarios adicionales

Mismo punto que lo anterior, solo que incluimos la capacidad de redirigir tanto salida como entrada para 1 comando. Si bien la entrada nos decantamos al final por la función "open" ante funciones como "fopen" y "freopen" debido a su simpleza. En la parte de salida no pudimos usar Open de nuevo ya que todos los requisitos de acceso que había que darle al archivo creaban interferencias entre ellos mientras que con la función "creat" no teníamos ningún problema.

2.3 Reconocer y ejecutar líneas con dos mandatos o mas

Ejemplo del comando: **ls | wc**

Ejemplo del comando: **ls | wc | sort**

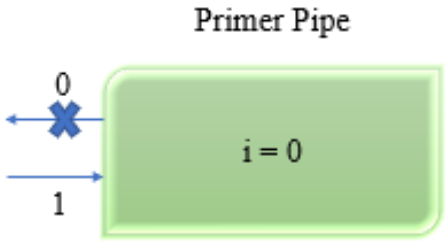
2.3.1 Implementación

Creamos una serie de tuberías en función de los mandatos que nos pasen (2 comandos o más). Para ellos nos valemos de la memoria dinámica ya que no sabemos cuántos comandos nos van a pasar.

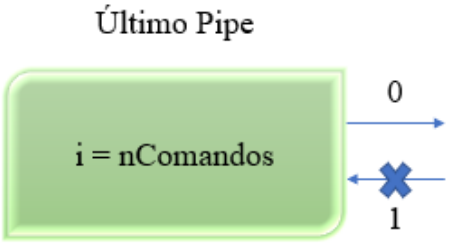
Al entrar ya al programa recorreremos todos los comandos y si son todos válidos entonces haremos fork(); como hicimos anteriormente en el ejecuta de 1 comando.

Aquí hay que diferenciar entre si es el primer hijo, el último o son hijos intermedios, ya que como se cierran los pipes y se comunican entre ellos es distinto para esos 3 casos.

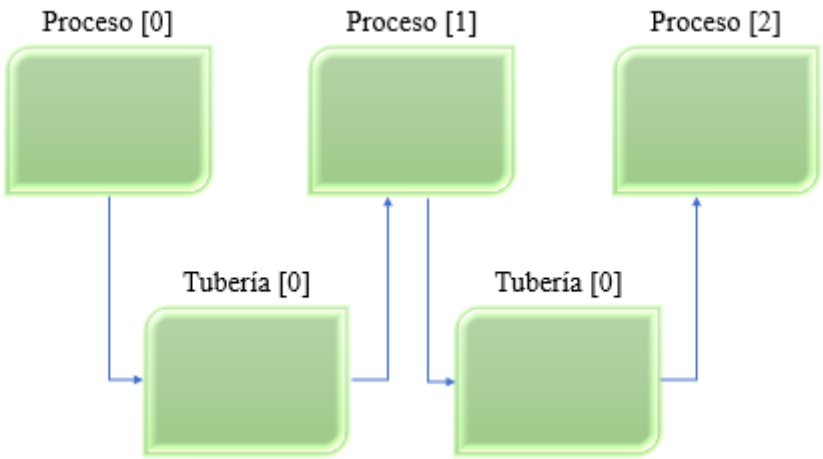
- Si es el primer hijo: De ser el primero, debemos cerrar la salida del pipe (el 0) y redirigir la salida estándar a la entrada de nuestro pipe. Por último, ejecutar con el comando execvp.



- Si es el último hijo: Se hará lo opuesto al caso anterior, o sea cerrar la entrada al pipe (el 1) y redirigir la salida de este hacia la entrada estándar, gracias a dup2.



- Si son hijos intermedios: Lo único que hay que hacer es comunicarlos entre sí, y cerrar las vías que no usemos. Así bien si en N dejamos la 1 abierta, cerramos la 0 y al contrario en su antecesor. Por último, ayudándonos de la salida y entrada estándar comunicamos ambos procesos entre sí.



2.3.2 Comentarios adicionales

En un principio creamos un subprograma que al ver que había 2 mandatos inicializaba una tubería para poder pasarse los datos entre ellos. Era tan simple como modificar el anterior “ejecuta” con un pipe. Esto se ve posteriormente que solo sirve como introducción, ya que no solo debería leer 2 mandatos sino más.
Seguimos conservando la capacidad de redirigir tanto de entrada como de salida, gracias a que lo hicimos como subprograma, al que llamamos “FuncionRedireccion”.

Posteriormente, tras numerosas dudas y decisiones erróneas y gracias a las explicaciones del profesor, decidimos implementar los pipes con memoria dinámica (malloc) y así poder crear cuantos sean necesarios en función de los comandos que tuviera la llamada (Obviamente tendremos N-1 tuberías, siendo N el total de los comandos, o sea 1 tubería por cada 2 comandos).
También nos dimos cuenta de que al hacer los hijos llamando a Fork(); deberíamos distinguir entre el primer hijo, el último y el segundo. Ya que el primer hijo debe cerrar la salida de la tubería, y el último lo contrario. Mientras que los hijos que estuvieran entre medias debían comunicarse entre ellos, así unos cerrarían la salida y el siguiente cerraría la entrada.
Las redirecciones para la entrada y salidas de los pipes hacia los estándares lo hicimos usando la función dup2.
Por último el tema de redirecciones queda igual que en los ejercicios anteriores gracias a que lo aplicamos como un subprograma. Lo único destacable en redirecciones es que nos hicimos valer de variables locales para volver a redirigir a las entradas estándares, después de haberlas modificado.

2.4 Capacidad para ejecutar el mandato cd

Ejemplo del comando: **cd “Mis Descargas”**

2.4.1 Implementación

A la hora de ejecutar los comandos, nuestra minishell mirará si es un comando cd, si ese es el caso, redijiremos el comando a un método nuestro creado aparte del resto, el cual, si le pasas más de 2 argumentos saltará un error, ya que no puedes hacer un cd a 2 directorios al mismo tiempo.
En el caso en el que solo detecte 1 el contador de argumentos de nuestro programa, el programa lo entenderá como que estás buscando salir al HOME.
Por último, si le pasarás 2 argumentos, trataríamos al segundo como la dirección, y a través de un printf junto con un getcwd mostraríamos al usuario donde se encuentra.

2.4.2 Comentarios adicionales

El CD le pasa como al apartado de 1 comando, usamos la estructura de un CD que ya habíamos hecho en el tema 3. Por lo que no supuso mayor complicación.
Tiene cosas interesantes como el uso del buffer al final para poder mostrar en que directorio nos encontrábamos. También lo añadimos como subprograma para que no se mezclara con el código de 1 comando y 2 comandos o más y así poder verlo más claro.

2.5 Señales desde teclado SIGINT y SIGQUIT

2.5.1 Implementación

Simplemente hemos usado el comando signal al principio de nuestro programa para inicializar las variables SIGINT y SIGQUIT.

2.5.2 Comentarios adicionales

Tras repasar las diapositivas nos dimos cuenta de que era algo bastante sencillo. Se resume en que mientras la minishell este activa sin ningún comando activo u hecho posteriormente ignore el Ctrl+C para salir.
Y que si esa en ejecución un comando o ya hicimos ejecución de algo previo si podamos usar Ctrl+C para salir de la Shell.
La implementación de las señales para el background no está ya que no terminados de hacer los comandos en background.