PRÁCTICA SISTEMAS OPERATIVOS

MYSHELL

Universidad Rey Juan Carlos

Álvaro Justo Rivas Alcobendas aj.rivas.2017@alumnos.urjc.es

Diego Pascual Ferrer d.pascual.2017@alumnos.urjc.es

INTRODUCCIÓN

La práctica pedida consiste en la creación de una terminal que reaccione ante los distintos comandos introducidos por teclado. A continuación, listaremos en rasgos generales las funciones que debe contemplar nuestra myShell:

-Mostrar un prompt.

-Leer líneas por teclado.

-Analizar las líneas usando la librería parser proporcionada para la práctica.

-Ejecución de los mandatos interpretados mediante la librería parser sobre las líneas obtenidas que se escribieron por teclado.

Ahora pasaremos a nombrar y explicar los archivos que conforman la Shell.

LÍBRERIA PARSER

Esta librería se forma mediante los dos archivos, libparser.a y parser.h .

Esta librería proporcionada para la práctica tiene como función principal la de interpretar los comandos que el usuario va introduciendo por teclado proporcionándonos una clase que contiene tanto los comandos, sus nombres, si contiene redirecciones de entrada y/o salida, errores.

De esta clase nosotros obtenemos los campos necesarios para interpretar todos los comandos introducidos, tanto si existen como si no.

MYSHELL.C

Este archivo contiene la codificación principal de todas las funciones y corresponde con la Shell en sí, esta codificada en C, el planteamiento de esta se ha basado en la memoria dinámica a la que recurrimos en la medida de lo posible para ahorrar espacio en memoria y ajustarnos en tiempo de ejecución a su ocupación siempre cambiante en referencia a lo introducido por el usuario.

Librerías usadas

A continuación, listaremos las librerías usadas y la razón de su uso:

Stdio.h: indispensable para la entrada/salida

Unistd.h: indispensable para el uso de procesos hijo y tuberías

Stdlib.h: permite reservar memoria dinámicamente.

Sys/wait.h: permite esperar procesos

Parser.h: esta es la librería explicada anteriormente proporcionada para la práctica.

String.h: nos permite comparar cadenas de caracteres.

Limits.h: útil para la reserva de memoria.

Sys/types.h: útil para conocer el tamaño de los elementos.

Sys/stat.h: permite crear directorios

Fcntl.h: útil para el tratamiento de procesos.

Errno.h: útil para el control de errores

Estructura principal

Después de la inclusión de las librerías, se inicia la codificación de la función main que contiene la declaración de las variables que hemos usado a lo largo del programa, que contiene: una variable que almacena matrices de descriptores de fichero, una variable que guarda los identificadores de los procesos creados, un buffer que usaremos para obtener los comandos leídos del teclado que los interpretara la librería parser, las señales, enteros para manejar el redireccionamiento de entrada y salida, una cadena de caracteres que utilizamos para almacenar la ruta de direccionamiento de donde se encuentra el usuario.

El siguiente apartado corresponde ya con las diversas funciones que, aunque algunas correspondan con mismas secciones las trataremos aparte para ser mas precisos, aunque incluiremos su localización y su explicación:

Realización de los comandos

Este bloque es el principal de toda la Shell, corresponde con el tratamiento de la clase proporcionada por la librería parser y la realización de los comandos.

Se inicia con un bucle en el que tratamos todos los comandos separados y en el que vamos realizando mediante la instrucción fork() y el pid correspondiente la creación de procesos hijos que mas tarde realizaran la ejecución del comando que les corresponde.

Dentro de este bucle encontramos dos bloques, la ejecución de un hijo cualquiera y la ejecución del padre que queda reservada únicamente a la asignación de los valores de los identificadores de los procesos hijos al array de identificadores; la ejecución del hijo se organiza en grandes rasgos generales basándonos en el comando que es: si es el primero, el último o un comando intermedio de la sucesión de comandos posibles introducidos en una línea.

En el primer comando tenemos un apartado dedicado a la redirección de entrada, otra a la de salida que explicaremos mas adelante; y el último apartado dedicado al cierre de tuberías que no estén en uso correspondiente, que son las tuberías que hay después de la primera, si la ejecución pertinente lo requería y la duplicación del descriptor de fichero que usamos para cambiar el redireccionamiento de la ejecución del comando que corresponda, si es el único comando se mantiene como hasta ahora escribiendo por pantalla el resultado, pero si hay mas comandos redirigiremos la salida de ese comando a la tubería por si el siguiente comando necesitara una entrada.

El siguiente apartado trata el comando final que contiene la redirección de salida en caso de ser necesaria y el duplicador del descriptor de fichero para obtener la entrada de la tubería anterior. Y el cerrado de tuberías no necesarias que corresponde con todas las tuberías anteriores menos la última de la que obtuvimos anteriormente la entrada.

El penúltimo apartado corresponde con la ejecución de un comando intermedio y por lo tanto el hijo intermedio al que corresponde. En este apartado engancharemos mediante dup2, como en los anteriores fragmentos, la entrada a la tubería con índice anterior y la salida a la tubería de índice. Posteriormente se cierran el resto de tuberías innecesarias.

En el último apartado del funcionamiento del hijo encontramos la instrucción execvp que ejecutara el comando correspondiente al hijo que este tratando, cabe destacar que la ejecución de los hijos se realiza en paralelo solamente que algunos al requerir de una entrada se quedaran esperando hasta obtenerla.

Cambio de directorio (cd)

Para la implementación de esta funcionalidad reutilizamos gran parte del código de la actividad de myCd. Está modificado para que se asemeje más al cd nativo de Linux. Además, para evitar que se realicen comandos otros comandos conectados con pipes se utiliza un controlador el cual:

- Si la salida de la función checkCommandsCD devuelve 0, no hay ningún cd en la lista de comandos a ejecutar.
- Si la salida de la función checkCommandsCD devuelve 1, ha habido un cd que se ha ejecutado y no ha habido ningún problema para ello.
- Si la salida de la función checkCommandsCD devuelve -1, ha habido un cd que no se ha ejecutado, ni lo harán el resto de los mandatos conectados mediante tuberías.

Redirecciones

En este apartado pasaremos a tratar los dos tipos de redirección presentes:

Redirección de entrada: esta redirección está presente solamente en el tratamiento del primer hijo ya que como pedía en la práctica esta solo se puede ejecutar en el primer comando, en esta redirección abrimos primero el fichero obtenido de la ruta proporcionada por el usuario, podría ser que este no existiera, pero eso lo explicaremos más adelante en el apartado de errores. En el caso de que si existiera procedería a situarse con el descriptor de fichero en el archivo para leerlo y obtener la entrada desde este punto redirigiendo la tubería de entrada del primer comando de la entrada estándar por defecto al archivo.

Redirección de salida: esta esta presente en dos segmentos en el primer hijo ya que podría ocurrir que fuera el primer comando y a la vez el único en esa línea y en la sección correspondiente al último comando, en ambas lo que realizamos es la apertura de un fichero comprobando si existe, si ya esta escrito lo truncaremos y si no existe se

creara en la ruta designada. Redirigiremos la salida mediante dup2 de la salida estándar por pantalla a este archivo.

Control de señales

Para controlar las señales SIGINT y SIGQUIT se utiliza un manejador de señales el cual, si alguna de las dos señales se emite mientras se está ejecutando un comando, se aplicará para todos los comandos que se mandó ejecutar. Por el contrario, si alguna de esas señales se emite mientras myShell está esperando comandos, será ignorada para evitar salir de myShell.

Manejo de errores

En este apartado explicaremos que utilizamos para el manejo de errores, utilizamos la librería errno.h que nos proporciona el tratamiento de errores asignando a cada error el número correspondiente de error y luego la función perror que imprime por pantalla una breve descripción del error al que corresponde, procedido del exit correspondiente en los hijos o en el caso del cd simplemente no se realiza ningún exit al no tener hijos.

Tenemos un manejo de error en la función fork por si falla en su funcionamiento, un manejo para los errores durante la entrada y la salida, dependiendo de la apertura de los ficheros, en el cd tratamos de que el archivo o directorio exista y la que la cantidad de argumentos no se exceda.

Prompt

El prompt que nosotros implementamos en el código aparece dos veces una antes de la primera ejecución y otra dentro del bucle de lectura, al final de este para que cada vez que se termine de ejecutar una línea vuelva a aparecer, contiene la ruta en la que nos encontramos y el msh para asemejarse lo máximo posible a la Shell original.

COMENTARIOS PERSONALES

La práctica en líneas generales nos ha parecido, aunque difícil bastante interesante, consideramos que contenía bastantes de los contenidos impartidos en clase, que nos ha permitido entender más a fondo los temas, desde los procesos que no entendimos hasta que no nos pusimos a realizar los ejercicios hasta las redirecciones que nos supusieron el menor problema dentro del programa.

Queremos destacar que tuvimos un error por desconocimiento que nos hizo perder bastantes días de trabajo dedicándonos a arreglar ese error y no pudiendo hacer el background por ello, el error correspondía con grep y las comillas que interpretábamos que no nos funcionaba el n comandos por esto hasta que descubrimos que no era problema nuestro sino de las comillas.