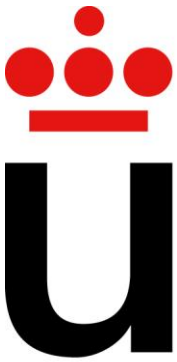

Universidad
Rey Juan Carlos

Sistemas Operativos

[PRÁCTICA 1 - MINISHELL]

IZAN CRUZ RUIZ Y SANTIAGO ANDRES LINARES OLIVIERIA

TABLA DE CONTENIDO

Autores

Los autores que hemos realizado esta práctica 1, que trata sobre el desarrollo de una minishell, somos **Izan Cruz Ruiz** y **Santiago Andrés Linares Oliveira**, dos alumnos que estudiamos el grado de ingeniería Informática en Vicálvaro, en el curso académico 2022-2023

Descripción del Código

Para poder montar y ejecutar correctamente esta práctica debemos de compilar mediante el comando **gcc**, nuestro archivo donde encontramos el código que explicaremos a continuación, y otro llamado archivo **libparser_64.a,** debido a que nuestro sistema Linux es x86_64, que contiene una función y dos tipos de datos, que nos son de gran ayuda en la construcción de la minishell.

Al comienzo del código, se incluyen todos los **#includes** necesarios para que las funciones que usamos dentro del programa funcionen de manera correcta. A continuación de ello incluimos las cabeceras de los distintos subprogramas, para que nuestro **main** funcione de una manera óptima. Dentro del **main()**, que es nuestro programa principal donde todo comienza, declaramos las variables y comenzamos a leer la línea que se nos introduce por pantalla, precedido siempre por el **msh>** especificado por los profesores para hacer la distinción de que nos encontramos ante nuestra minishell. Donde debemos diferenciar entre si la línea es distinta de null, en la que se ejecutará la función **ejecuta_comandos()**, que se explicará más tarde o si la línea contiene el "&", símbolo característico para especifficar que se ejecutará en segundo plano. También debemos de mencionar que hemos incluido antes del while para la lectura de la línea y después de terminar hemos incluido la función **signal(SIGINT, SIG_IGN)** con el objetivo de ignorar las señales que se introducen por teclado y que el programa muera.

La otra gran función que encontramos en nuestro código es el **ejecuta_comandos()**, donde debemos diferenciar entre los comandos que se insertan por la línea que luego leerá el main, debido a que son las dos grandes ramificaciones del programa. En el caso de que solo haya una única opción, hemos incorporado mediante la función **strcmp()**, que principalmente se encarga de comparar cadenas de strings, que si hay coincidencia ejecute el programa correspondiente. Por ello encontraremos la opción de que se ejecute **comandocd()**, donde se ejecutará la función cd, donde si el argumento que sigue a cd es nulo, nos llevará directamente a **home (home/usuario),** mientras que se si le pasa un argumento, comprobará la existencia de ese directorio y nos mostrará un mensaje de cambio en caso de éxito o error. Otra opción que encontramos es **exit()**, que si lo ejecutamos nos sacará directamente de la minishell. La ultima opción que encontramos es el **comandoumask()**, que nos devolverá la mascará de lo último ejecutado.

Si al insertar el comando no se corresponde con ninguno de los mencionados, se ejecutará el comando mediante la función **excecvp()**, y para que termine la función deberemos esperar a recibir la señal del hijo, por ello ponemos el **wait()**, y si recibimos una señal distinta sabremos que el comando no se habrá ejecutado correctamente.

Ahora encontramos la otra parte de la función de ejecuta_**comandos()**, en la que recibimos más de un argumento separado por tuberías, por ello creamos un array bidimensional dado que habrá que diferenciar entre **cada pipe** y sus partes de **lectura [0] y escritura [1]**. Dado que cada array leerá la parte de lectura del pipe anterior, por ello debemos distinguir entre tres momentos, entre el primer mandato, donde haremos uso de la función **dup2** que nos ayuda a crear una copia del descriptor, para que la lectura sea mucho más sencilla y eficiente, e igual sucede con el último mandato, solo que debe de leer del i-1, mientras que los mandatos intermedios, deben de leer del sucesor. Terminado esto cerraríamos todos los pipes con **close(p)**, para que no quedase ninguno abierto, ejecutaríamos el comando y seguiríamos el mismo proceso que para un único mandato con el padre, terminando con la liberación de memoria de estos pipes, con **free()**. Y como hemos hecho en el main, también hemos incluido la función de signal.

Por último encontramos la función de las redirecciones, donde asignaremos a una variable el valor que obtenemos después de ejecutar el comando **open()**, que se centra en poder abrir o cerra ficheros, y que tiene unas banderas que nos ayudan a indicar la forma de apertura y por el último encontramos una variable opcional que especifica los permisos de los archivos, y si recibimos el valor de -1, sabremos que ha sucedido un error al ejecutar este comando, en caso de éxito, pasaríamos al comando **dup2**, donde redijéremos la entrada o salida en función del tipo de redirección (entrada, salida y error) a un archivo que hemos especificado.

• Diseño del Código

Las variables globales que hemos usado en este trabajo de la minishell son el tline y el line, presentes en todos los programas que hemos desarrollado, estas son variables que se utilizan para almacenar la línea de comandos que se ingresa en la consola y parsearla para ejecutar el comando correspondiente, en nuestro comando el comando excecvp

Variables globales	Tipo	Descripción
Tline	**Char**	**Su función principal es almacenar toda la línea de comandos completa**
Line	**Char**	**Su función principal es almacenar cada uno de los argumentos individuales que se insertan en la línea.**
Buffer	**Char**	**Su función principal es almacenar los comandos que el usuario introduce**

El algoritmo utilizado en nuestra minishell sigue un claro algoritmo donde lo primero que hacemos es leer la línea de comandos que se inserta por pantalla, y en función de lo que haya escrito el usuario le damos una finalidad u otra, es decir, se ejecuta el comando o, por el contrario, sino es conocido por la máquina, mostrará un mensaje de error.
La estrategia utilizada al ejecutar los mandatos ha sido principalmente distinguir entre el número de comandos que se pasan por pantalla, por ello debemos diferenciar si únicamente se ejecuta un único comando como cd, exit … Y por el contrario si encontramos en ncommands más de un argumento, donde entran en juego los pipes, en los que seguimos la siguiente estructura. Primero construimos el array bidimensional, ya que habrá n tuberías y cada tubería está compuesta por dos extremos uno de lectura [0] y otro de escritura [1], iniciamos el bucle for de lectura y vamos leyendo, y si no encontramos fallos al hacer el comando dup2, es decir, no devuelve -1, redijéremos la entrada, al terminar de leer todos los comandos cerramos todos los pipes, y ejecutamos la línea, para luego liberar el espacio con free.
La implementación de CD sigue también el mismo patrón, comprobamos que la línea no sea nula, y asignamos una variable al comando open, ya que nos devolverá un valor, en el que comprobaremos si el numero es mayor de 0, ya que significa éxito, es decir, si es mayor, se ejecutará y cambiará de directorio, si no mostrará un mensaje de error. En caso de que la línea sea nula, se devolverá a home (home/usuario), dado que ha así ha sido asignado a la variable home.
El comando exit, se ejecuta comprobando mediante el comando strcmp, si la cadena introducida es exit, si es así, nos sacará de la minishell. El comando umask, comprobamos si la línea no es nula, recuperamos el valor y lo imprimimos, gracias a al comando umask.
En cuanto a las señales, hemos seguido un patrón similar al comando cd, donde dirigimos la línea hacia la información de redirección de entrada de una línea de comandos. Y asignaremos a la variable rd, el número resultante de hacer el comando open, comprobando si se realiza con éxito o no, si es el caso afirmativo redijéremos hacia el fichero indicado en el rd.

• Principales Funciones

1. MAIN

	Nombre Función	Nombre	Tipo	Descripción
Argumentos	Argumento 1	**Argc**	**Int**	**Entero que indica cuantos argumentos recibe el main**
	Argumento 2	**Argv[]**	**Char**	**Cadena de caracteres que recibe los argumentos que recibe el main**
Variables Locales	Variable 1	***Line**	**TLine**	**Línea de comandos que el usuario pasa por pantalla**
Descripción de la Función	**En función de lo que se pase el programa ejecutará una ramificación u otra, si se le pasa un comando ejecutara el comando, también validará si está o no en segundo plano (&)**			

2. Ejecuta Comandos

	Nombre Función	Nombre	Tipo	Descripción
Argumentos	Argumento 1	***Line**	**TLine**	**Línea de comandos que el usuario pasa por pantalla**
Variables Locales	Variable 1	**i**	**int**	**Ayuda auxiliar para la creación del array bidimensional de los pipes.**
		j	**int**	**Ayuda auxiliar con la principal función de hacer el close en los pipes**
		k	**int**	**Ayuda auxiliar con la principal función de hacer el close en los pipes**
		** **p**	**int**	**Puntero a puntero, que apunta a la dirección de memoria donde esta creado el array bidimensional**
		status	**int**	**Variable que devuelve el estado del comando, 0 ha sido correcto, mientras que si es distinto de 0 incorrecto.**
		pid	**Pid_t**	**Almacenar el pid del proceso hijo.**

Descripción de la Función	**Ejecuta los comandos que se le pasan al main, en función de si recibe uno o varios argumentos**

3. Redirecciones

	Nombre Función	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	***Line**	**TLine**	**Línea de comandos que el usuario pasa por pantalla**
Variables Locales	Variable 1	**rd**	**int**	**Ayuda auxiliar para guardar el número devuelto al hacer la ejecución del mandato rd, ya que -1 es erroneo y distinto de él es acierto.**
Descripción de la Función	**Crea las redirecciones o redirige en función de la existencia o no de los ficheros. Se especifican tres tipos de redirección: Entrada, Salida y Error.**			

4. Comando Umask

	Nombre Función	**Nombre**	**Tipo**	**Descripción**
Argumentos	Argumento 1	***Line**	**TLine**	**Línea de comandos que el usuario pasa por pantalla**
Variables Locales	Variable 1	**mask**	**Mode_t**	**Variable auxiliar para guardar la máscara de permisos numéricos, estableciendo los permisos que tiene ese archivo al crearse o abrirse**
Descripción de la Función	**Devuelve el valor de la mascará numérica, especificado en ese momento, donde también existe la posibilidad de poder cambiarlo.**			

5. Ignorar Signt

	Nombre Función	Nombre	Tipo	Descripción
Argumentos	Argumento 1			
Variables Locales	Variable 1			
Descripción de la Función	**Sirve para incluir un salto de línea al hacer Ctrl + C**			

6. Comando CD

	Nombre Función	Nombre	Tipo	Descripción
Argumentos	Argumento 1	***Line**	**TLine**	**Línea de comandos que el usuario pasa por pantalla**
Variables Locales	Variable 1	***** **Home**	**Char**	**Puntero que apunta a la variable de entorno especificada, en este caso HOME**
		Dr	**Int**	**Sirve para guardar el valor de hacer el comando chdir, donde si es mayor que 0 es error y si eso 0 es éxito.**
Descripción de la Función	**La principal función es cambiar de directorio al especificado, en caso de no especificar directorio cambiará a HOME (home/usuario).**			

- # Casos de Prueba

Caso de prueba cd y exit, vemos que se ejecuta correctamente

```
izancruz@izancruz-VirtualBox:~/minishell$ gcc -Wall -Wshadow -o minishell_ICR_SALO minishell_ICR
_SALO.c libparser_64.a -static
izancruz@izancruz-VirtualBox:~/minishell$ ./minishell_ICR_SALO
msh>$ cd
Cambiando a directorio: /home/izancruz
msh>$ pwd
/home/izancruz
msh>$ exit
izancruz@izancruz-VirtualBox:~/minishell$
```

Caso de Prueba con pipes y redirecciones.

```
izancruz@izancruz-VirtualBox:~/minishell$ ./minishell_ICR_SALO
msh>$ ls -l | tr "r" "T" | wc -l
18
msh>$ ls -l | tr "r" "T" > ficheroprueba.txt
msh>$ cat ficheroprueba.txt
total 3752
-Tw-Tw-T-- 1 izancTuz izancTuz      0 dic 12 23:17 ficheTopTueba.txt
-Tw-Tw-T-- 1 izancTuz izancTuz  21802 nov 11 10:19 libpaTseT_64.a
-Tw-Tw-T-- 1 izancTuz izancTuz  13054 nov 11 10:22 libpaTseT.a
-TwxTwxT-x 1 izancTuz izancTuz 931136 nov 30 10:29 minishell
-Tw-Tw-T-- 1 izancTuz izancTuz   9506 nov 30 10:17 minishell.c
-TwxTwxT-x 1 izancTuz izancTuz 926392 dic  5 21:03 minishelldef
-TwxTw-T-- 1 izancTuz izancTuz   2645 dic  5 21:27 minishelldef.c
-TwxTwxT-x 1 izancTuz izancTuz 931232 dic 12 23:13 minishell_ICR_SALO
-Tw-Tw-T-- 1 izancTuz izancTuz   7450 dic 12 21:28 minishell_ICR_SALO.c
-TwxTwxT-x 1 izancTuz izancTuz 931224 dic 12 21:07 minishellpTuebas
-Tw-Tw-T-- 1 izancTuz izancTuz   7437 dic 12 21:06 minishellpTuebas.c
-Tw-Tw-T-- 1 izancTuz izancTuz    281 nov 11 10:15 paTseT.h
-Tw-Tw-T-- 1 izancTuz izancTuz  13451 nov 30 10:22 PTactica_minishell_apoyo.zip
-Tw-Tw-T-- 1 izancTuz izancTuz    855 dic  8 21:09 pTueba1.txt
-Tw-Tw-T-- 1 izancTuz izancTuz    918 dic  8 21:13 pTueba2.txt
-Tw-Tw-T-- 1 izancTuz izancTuz    792 dic  8 19:07 pTueba.txt
-Tw-Tw-T-- 1 izancTuz izancTuz      0 dic  9 13:19 Tambo.txt
-Tw-Tw-T-- 1 izancTuz izancTuz    872 nov 11 10:16 test.c
msh>$
```

Caso de prueba de umask y el cambio de máscara

```
msh>$ umask
Máscara de permisos numérica: 0002
msh>$ umask 777
msh>$ umask
Máscara de permisos numérica: 0777
msh>$ exit
izancruz@izancruz-VirtualBox:~/minishell$
```

Comentarios Personales

Esta práctica nos ha servido para mejorar nuestros conocimientos sobre el lenguaje de programación y la cantidad de funciones que se pueden usar dentro de ella, algo que sin duda nos servirá para nuestro futuro laboral. Nos hemos encontrado con problemas a la realización de los pipes y para conseguir implementar las señales, que hacen que el programa no se muera. También hemos encontrado bastantes dificultades para elaborar tanto el Jobs como el fg, dado que sabíamos la base sobre la que partir ya que debíamos crear un tipo estructurado que nos permitiese poder ir metiendo aquellos procesos que se ejecutan en background, para luego hacer el jobs, que básicamente es hacer un bucle para mostrar esa lista de tareas, y el fg consistiría en recibir dos argumentos, uno el comando fg y el otro el número del comando que desea ejecutar en primer plano, por ello el padre esperaría mediante el wait, a que su hijo realizase la tarea correspondiente.

Como mejora plantearía en las clases, dedicar alguna clase a explicar los conocimientos principales de la práctica, sabemos que existen cierta similitud con los ejercicios planteados, pero luego son cosas distintas, dado que son situaciones que nunca nos habíamos planteado y conceptualmente son complejas. Hemos dedicado 2h semanales desde la publicación de la práctica y hemos terminado de afianzar el diseño y los demás conceptos durante la última semana, si es cierto que podríamos haber dedicado mucho más tiempo, intentaremos solucionarlo para la siguiente práctica.