MiniShell

PRÁCTICA 1

Sistemas Operativos – Ingeniería Informática

Iván Conde Carretero – 039402137

Lara Camila Sánchez Correa – Y0479214T

Tabla de contenido

En el siguiente documento explicaremos el desarrollo de la primera práctica de la asignatura: myShell. Pasando por distintos aspectos, tanto como su funcionalidad, corrección y la elección de los distintos algoritmos.

En primer lugar, hablaremos de la autoría.

Autoría

La siguiente práctica ha sido desarrollada por dos alumnos del Grado en Ingeniería Informática de Móstoles del tercer curso.

Iván Conde Carretero – 03940213Z – Expediente: 1497

Lara Camila Sánchez Correa – Y0479214T – Expediente: 1520

La totalidad de la práctica ha sido especificada por ambos estudiantes.

A continuación, se explicará el código elegido.

Descripción del código

Podremos separar el código presentado según la funcionalidad de cada aspecto. A grandes rasgos nos encontramos con los siguientes apartados:

Main

En esta sección se realizará la parte más importante de la práctica. Como el desarrollo del comando interno CD, así como el Jobs, la ejecución de un solo comando, las redirecciones estándar y por último la llamada al método para ejecutar varios comandos.

Mandato CD

Empezando por el comando CD, se ha tenido que desarrollar el código entero en la práctica, ya que al ser un mandato interno se debía ejecutar en la totalidad de la *myShell*.

El comando CD, realiza las mismas operaciones que su análogo en la terminal. Podremos comprobarlo en la siguiente imagen.

```
msh>cd
msh>pwd
/home/alumno
msh>cd Escritorio/SSO
Escritorio/SSOmsh>pwd
/home/alumno/Escritorio/SSO
msh>
```

Como vemos, al cambiar el directorio sin ningún argumento más, cambiará al directorio HOME obtenido mediante la función *getenv*(). Al cambiar por alguna ruta, cambiará de forma correcta mediante *chdir*("*ruta*"). Y si el mandato recibe alguna entrada que no reconoce, dará el siguiente error:

```
Error al cambiar de directorio
```

Mandato Jobs y FG

Para la realización de estos dos mandatos, hemos creado una lista simple enlazada (programada por nosotros mismos desde cero) para almacenar el nodo que se ejecuta en background, dicho nodo contiene la información de identificador del proceso, el estado y el valor del Jobs que sirve como identificador.

Nuestra lista tiene los métodos *addLast* añade al final para añadir un nuevo proceso en Background, *removeLast* elimina el último proceso de la lista, *remove_Link* elimina un proceso concreto, *imprimirJobs* nos muestra en pantalla todo los procesos que se están ejecutando o parados ,y *getPid* o *getLastPid* que nos devuelve el identificador del proceso.

Nuestro mandato fg se puede ejecutar pasándole el identificador del Jobs o sin identificador. Si no se le pasa ningún identificador, cogerá el id del último nodo de la lista, eliminará dicho nodo y esperara hasta que el proceso acabe (waitpid(pidProceso, NULL, 0), en el caso contrario cogeremos el identificador del nodo cuyo valor Jobs se le pase, borraremos dicho nodo y esperaremos hasta que el proceso acabe o mande una señal.

```
msh> xman &
[1]  14862
msh>Warning: Cannot convert string "-*-helvetica-medium-r-*-*-*-120-*-*-*-*-*-*" to type FontStruct
xman &
[2]  14863
msh>Warning: Cannot convert string "-*-helvetica-medium-r-*-*-*-120-*-*-*-*-*-*" to type FontStruct
jobs
[1]   14862   Ejecutando     xman &

[2]   14863   Ejecutando     xman &

msh>fg
^Cmsh>
msh>jobs
[1]   14862   Ejecutando     xman &

msh>fg
^Cmsh>
msh>jobs
msh>
```

Como se muestra en la foto, se realiza dos mandatos en background, se vuelven a ejecutar en y los eliminamos mediante la señal SIGINT (CTRL + C).

También tenemos que destacar que hemos creado una función *comprueba(),* que cada vez que lee una línea la myShell, comprobara si hay algún proceso que ya haya acabado y si es así lo eliminara de la lista.

Mandatos externos

La ejecución de un solo mandato también se ha declarado en el *main*. Se han tenido en cuenta los errores que puedan producir los *fork,* así como las redirecciones estándar.

Vemos un ejemplo de mandato:

```
msh>ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::e3e7:63c8:b7f5:ea61  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:7e:c3:11  txqueuelen 1000  (Ethernet)
        RX packets 2828  bytes 3566775 (3.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 651  bytes 82644 (82.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Bucle local)
        RX packets 106  bytes 9601 (9.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 106  bytes 9601 (9.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

msh>
```

Lo ejecuta correctamente y a su vez, volverá a mostrar el *prompt* una vez ejecutado.

Redirecciones:

```
msh>cat > texto.txt
Bienvenido a nuestra shell
^Cmsh>cat < texto.txt
Bienvenido a nuestra shell
```

Ejecución de varios mandatos

Antes se ha mostrado el comando ifconfig, en esta ocasión mostraremos le comando ifconfig con redirección de salida al comando Word count.

```
msh>ifconfig | wc
     18     108     871
msh>
```

A continuación, mostraremos varios comandos con varios pipes:

El desarrollo del código de varios comandos se ha realizado en un método a parte para mejorar la comprensión del código.

Podemos ver que se ha utilizado un array de tuberías con n-1 comandos, es decir, si entran 3 comandos por consola conectados por pipes, se utilizarán 3-1 = 2 pipes en total.

Así como la creación de un array de Foreground, para saber si se realiza la tarea en Fg o Bg, ya que tienen tratamientos diferentes, uno deberá esperar las señales.

Se han tenido en cuenta los errores que se puedan provocar en la creación de subprocesos hijos y también se han cerrado los pipes que no se vayan a utilizar en cada caso para hacer un buen uso de estos.

Tendremos tres opciones a la hora de enfrentarnos a un subproceso creado, que sea el primer mandato, un mandato intermedio o el último. Cada uno se desarrolla de distinta manera, ya que deberán cerrar las entradas/salidas de los pipes que no utilicen.

Se puede apreciar de manera más gráfica en el siguiente esquema:

[Cerrará en el pipe la lectura] **Comando Primero** **//Dado que SOLO va a escribir**

Pipe

Comandos Intermedios

Pipe

//Dado que SOLO va a leer **Comando último** **[Cerrará en el pipe la escritura]**

Finalmente se liberará la memoria dinámica reservada para cada array descrito, y cerrar los pipes que pueden haber quedado abiertos.

Foreground y Background

En nuestra myShell hemos implementado procesos foreground y backgroud, la diferencia entre estos dos es a la hora de hacer la espera, ya que cuando se ejecuta un comando en foreground deberá de esperar a que termine el proceso para poder ejecutar el siguiente,

en el caso contrario no habrá que esperar para un siguiente proceso, así que se seguirán ejecutando los procesos. Para la realización del background hemos realizado el waitpid(pidProceso, NULL, WNOHANG), permitiendo la continuación del programa y dejando ejecutar nuevos mandatos, en decir, no sé bloquea.

```
msh>xman
Warning: Cannot convert string "-*-helvetica-medium-r-*-*-*-120-*-*-*-*-*-*" to type FontStruct
sdf
fsdf
sdf
sdfsdf
sdfss
^Cmsh>
```

Como se puede mostrar en la imagen superior hemos ejecutado un comando en foreground y aunque sigamos escribiendo en nuestra myShell no se ejecutara ningún mandato, ya que el foreground hace la espera waitpid(pidProceso, NULL, 0), por lo que bloquea y no nos dejara continuar hasta que se acabe el proceso o usemos alguna señal para finalizar.

En la siguiente imagen mostramos un ejemplo de ejecutar un mandato en Background, nos devolverá el id del proceso y a continuación podremos ejecutar nuevos mandatos.

```
msh>xman &
[1]  15272
```

Funcionalidades

La práctica también cuenta con la redirección de la entrada y salida estándar. Así como, la correcta implementación de las librerías dadas, lectura de una línea de teclado de uno o varios mandatos y así como la creación de varios subprocesos hijos con cada uno de ellos.

Puntos claves, la myShell mostrará un *prompt* e ignorará las señales especificadas en la práctica:

```
msh>
msh>
msh>^C^C^C^C^C^C^C
msh>^\^\^\^\^\^\^\
msh>
msh>
```

Dichas señales realizan el objetivo esperado, se bloquean correctamente para no salir de la myShell o en procesos en Background, esto es así porque hemos usado SIG_IGN para ignorar la señal, por lo que dicha señal ha sido colacada al principio de nuestro programa y en background.

Por lo contrario, se ha programado dos manejadores, ambos obtenían el identificador del proceso mediante la variable státic pidForeground, y ambos manejadores se activaban según la señal.

El primer manejador se activaba mediante CTRL + C y mataba mediante la función kill, cuyos argumentos son pidForeground y SIGTERM, termina el proceso de una forma no violenta.

El segundo manejador se activa mediante CTRL + \ y mata el proceso mediante la función kill, con lo argumentos pidForeground y SIGKILL, en este caso hemos usado SIGKILL, debido que termina el proceso de forma *abrupta,* no puede ser ignorada y el objetivo principal es que por defecto genera un fichero ***core*** al terminar el proceso.

Es capaz de leer una línea de teclado y así como esperar si se ha dado al enter.

Analiza la librería *parser* gracias a la sucesión de comandos:

```
gcc myshell.c libparser.a -o test -static
```

Conclusión

En esta práctica se ha podido ver de primera mano como realizar código en C, siendo un gran puente para más lenguajes de programación como C++ o incluso C#.

También se ha podido explorar con el entorno de Linux, ver sus pros y contras ante otros sistemas operativos y descubrir otro tipo sistemas.

Alguno de los problemas que se han encontrado durante el desarrollo de la práctica ha sido el uso de las librerías Parser, que una vez bien implementadas han hecho una labor increíble para el desarrollo, sin embargo, antes de llegar a ese punto se probaron en varios entornos de desarrollo (no la propia terminal de Linux) y en ninguno se consiguió exportar de forma correcta las librerías. Finalmente, se pudo lograr utilizando el mandato -static recomendado por los profesores.

Otro de los problemas ha sido intentar conseguir el pleno funcionamiento de Background, pero no hemos llegado al objetivo, ya que hemos tenido grandes dificultades en nuestra estructura de datos, ya que no realizaba correctamente el funcionamiento que esperábamos. Nuestra idea era recorrer la lista de procesos cada vez que se leía una línea por entrada y comprobar si había algún proceso acabado para matar a los hijos y no quedarse con procesos zombies.

El ultimo problema fue en el desarrollo de las señales. Nuestras señales realizan el objetivo de la práctica, pero cuando estamos en foreground con un proceso y realizamos CTRL + C

o CTRL + \, realiza la función del manejador correctamente pero nos muestra por pantalla CTRL + C o CTRL + \ más el promp.

Como crítica se apreciaría tener más práctica variada en el entorno de C, ya que, se ha tenido que recurrir muchísimo a la investigación por cuenta propia para entender bien el funcionamiento de algunas características.

Un punto de mejora para nuestra práctica sería incluir todos las redirecciones de fichero a un mismo método, para no tener que hacerlas dos veces (una para un solo comando y otra para varios comandos).

Finalmente, se comentará el tiempo dedicado a la práctica. De primera mano se tenía con firmeza la estructura que se implementaría, sin embargo, conseguir que dicha estructura funcionase fue la parte más tediosa y costosa en tiempo. Una vez que se consiguió que la estructura troncal compilase empezamos a desarrollar el resto del código a partir de ella, como los comandos internos y el foreground/background. Por tanto, se cree que el tiempo dedicado a la primera parte fue el necesario, ya que gracias a ello se pudo desarrollar la práctica en el tiempo justo.