

Práctica obligatoria 1

MiniShell

Descripción

En esta práctica se abordará el problema de implementar un programa que actúe como intérprete de mandatos. El *minishell* a implementar debe interpretar y ejecutar mandatos leyéndolos de la entrada estándar. En definitiva debe ser capaz de:

- Ejecutar una secuencia de uno o varios mandatos separados por el carácter '|’.
- Permitir redirecciones:
 - Entrada: < **fichero**. Sólo puede realizarse sobre el primer mandato del *pipe*.
 - Salida: > **fichero**. Sólo puede realizarse sobre el último mandato del *pipe*.
 - Error: >& **fichero**. Sólo puede realizarse sobre el último mandato del *pipe*.
- Permitir la ejecución en *background* de la secuencia de mandatos si termina con el carácter '&’. Para ello, el *minishell* debe mostrar el **pid** del proceso por el que espera entre corchetes, y no bloquearse por la ejecución de dicho mandato (es decir no debe esperar a mostrar el *prompt* a su terminación).

A grandes rasgos, el programa tiene que hacer lo siguiente:

- Mostrar en pantalla un *prompt* (los símbolos **msh>** seguidos de un espacio).
- Leer una línea del teclado.
- Analizarla utilizando la librería *parser* (ver apéndice).
- Ejecutar todos los mandatos de la línea a la vez creando varios procesos hijo y comunicando unos con otros con las tuberías que sean necesarias, y realizando las redirecciones que sean necesarias. En caso de que no se ejecute en *background*, se espera a que todos los mandatos hayan finalizado para volver a mostrar el *prompt* y repetir el proceso.

Se tendrá en cuenta lo siguiente:

- Si la línea introducida no contiene ningún mandato o se ejecuta el mandato en *background*, se volverá a mostrar el *prompt* a la espera de una nueva línea.
- Si alguno de los mandatos a ejecutar no existe, el programa debe mostrar el error “**mandato: No se encuentra el mandato**”.
- Si se produce algún error al abrir cualquiera de los ficheros de las redirecciones, debe mostrarse el error “**fichero: Error. Descripción del error**”.
- Ni el *minishell* ni los procesos en *background* deben finalizar al recibir la señal desde teclado **SIGINT (Ctrl-C)** mientras que los procesos que se lancen deben actuar ante ellas, manteniendo la acción por defecto.

Objetivos parciales

- Ser capaz de reconocer y ejecutar en *foreground* líneas con un solo mandato y 0 o más argumentos. **(0.5 puntos)**
- Ser capaz de reconocer y ejecutar en *foreground* líneas con un solo mandato y 0 o más argumentos, redirección de entrada estándar desde archivo y redirección de salida a archivo. **(1 punto)**
- Ser capaz de reconocer y ejecutar en *foreground* líneas con dos mandatos con sus respectivos argumentos, enlazados con '|’, y posible redirección de entrada estándar desde archivo y redirección de salida a archivo. **(1 punto)**
- Ser capaz de reconocer y ejecutar en *foreground* líneas con más de dos mandatos con sus respectivos argumentos, enlazados con '|’, redirección de entrada estándar desde archivo y redirección de salida a archivo. **(2,5 puntos)**

- Ser capaz de ejecutar el mandato **cd** (0,5 puntos). Mientras que la mayoría de los mandatos son programas del sistema, **cd** es un mandato interno que debe ofrecer el propio intérprete de mandatos. El mandato **cd** debe permitir el acceso a través de rutas absolutas y relativas, además de la posibilidad de acceder al directorio especificado en la variable **HOME** si no recibe ningún argumento, escribiendo la ruta absoluta del nuevo directorio actual de trabajo. Para el correcto cambio de directorio el mandato **cd** se debe ejecutar sin *pipes*.
- Ser capaz de ejecutar el mandato interno **exit** (0,5 puntos). Este mandato servirá para abandonar el intérprete de mandatos.
- Ser capaz de ejecutar el mandato interno **umask** (1 punto). Este mandato controla la máscara con los permisos de creación de nuevos ficheros. Cuando **umask** imprima los permisos por la salida estándar podrá hacerlo en forma numérica o simbólica, dependiendo de las opciones de llamada. Sin embargo, bastará con admitir el parámetro *mode* únicamente en formato octal (con tres dígitos octales, o con un 0 seguido de tres dígitos octales; en todo caso, los tres últimos dígitos representan la máscara). Recuérdese que el mandato **umask** emplea la forma octal para especificar los permisos que **no** se establecerán, y la forma simbólica para indicar los permisos que **sí** se establecerán.
- Ser capaz de reconocer y ejecutar tanto en *foreground* como en *background* líneas con más de dos mandatos con sus argumentos, enlazados con '|', con redirección de entrada estándar desde archivo y redirección de salida a archivo. Para su correcta demostración, se deben realizar los mandatos internos **jobs** y **fg** (2 puntos):
 - **jobs**: Muestra la lista de procesos que se están ejecutando en segundo plano en la *minishell* (no es necesario considerar aquellos procesos pausados con **Ctrl-D**). El formato de salida será similar al del mandato **jobs** del sistema:


```
[1]+  Running                  find / -name hola | grep h &
```
 - **fg**: Reanuda la ejecución del proceso en *background* identificado por el número obtenido en el mandato **jobs**, indicando el mandato que se está ejecutando. Reorienta su entrada estándar y salidas estándar a la terminal del usuario. Si no se le pasa ningún identificador se pasa a *foreground* el último mandato en *background* introducido.
- Evitar que los mandatos lanzados en *background* y el *minishell* mueran al enviar desde el teclado la señal **SIGINT**, mientras los procesos en *foreground* responderán ante ellas. El comportamiento del intérprete de mandatos al pulsar **Ctrl-C** deberá ser similar al de **bash**. (1 punto)

Nota: las puntuaciones para cada objetivo parcial son las puntuaciones **máximas** que se pueden obtener si se cumplen esos objetivos. No se debe hacer un programa separado para cada objetivo, sino un único programa genérico que cumpla con todos los objetivos simultáneamente.

Normas de estilo

La práctica se evaluará comprobando el correcto funcionamiento de los distintos objetivos, y valorando la simplicidad del código, los comentarios, la óptima gestión de recursos, la gestión de errores y la calidad de la memoria. El profesor podrá solicitar una defensa oral de la práctica si lo considerase necesario.

A la hora de codificar la solución pedida, se deberán respetar una serie de normas de estilo:

- Las variables locales de las funciones se declararán inmediatamente después de la llave de comienzo del cuerpo de la misma. Se penalizará la declaración de variables entre sentencias ejecutables de las funciones.
- No se admitirán asignaciones iniciales a variables cuyo valor dependa del valor de otras variables. El valor asignado en estos casos siempre deberá ser conocido en tiempo de compilación.
- Cuando se declare una variable de tipo *array*, su tamaño deberá ser conocido en tiempo de compilación. Los *arrays* de tamaño variable se crearán en memoria dinámica mediante las funciones correspondientes (**malloc**, **calloc** o **realloc**). La memoria dinámica solicitada deberá liberarse (**free**) antes de salir del programa.
- Las operaciones sobre *strings* (copia, comparación, duplicación, concatenación, etc) se realizarán en lo posible mediante las funciones indicadas en *string.h* (ver referencias a la biblioteca de C en el Campus Virtual).
- En general, se penalizará el uso de construcciones propias de C++.
- Al compilar el código fuente, deberá producirse el menor número posible de *warnings*. Lo más correcto sería que no se produjese ninguno.

El incumplimiento de estas normas de estilo, así como de otras normas que puedan ser anunciadas por el equipo docente a través del Campus Virtual, conllevará una penalización en la nota.

Entrega de prácticas

La entrega de prácticas se hará a través del Campus Virtual en las fechas anunciadas en el mismo. Se debe entregar un único archivo *myshell.c* con el código de toda la práctica, debidamente comentado, y una memoria en formato **PDF**. La memoria debe realizarse a partir de la plantilla Word aportada junto al enunciado, y debe incluir:

- Una portada con el nombre de los autores.
- Un índice de contenidos.
- Una descripción del código, incluyendo descripción de las principales funciones implementadas.
- Comentarios personales: incluyendo problemas encontrados, críticas constructiva, propuesta de mejoras y evaluación del tiempo dedicado.
- NO INCLUIR CÓDIGO FUENTE.
- NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA. Aprobar la memoria es tan imprescindible para aprobar la práctica como el correcto funcionamiento de la misma. Si al evaluar la memoria se considera que no alcanza el mínimo admisible, la práctica se considerará SUSPENSA.

IMPORTANTE: Todos los archivos pedidos (fuentes y memoria) se empaquetarán en un único archivo comprimido en formato ZIP llamado **practica1.zip** y entregarse en Aula Virtual pinchando en el enlace correspondiente.

Defensa oral de la práctica

Después de entregar la práctica, cada alumno será convocado de forma **individual** para realizar una defensa oral de su práctica, en fecha y lugar indicados por el equipo docente de la asignatura.

En la defensa oral de la práctica, el alumno:

- Compilará y montará el ejecutable de su práctica desde la línea de mandatos de Linux.
- Ejecutará y demostrará el funcionamiento de su práctica, de acuerdo con las indicaciones del profesor.
- Responderá a cuestiones relativas al código fuente de su práctica.

La defensa oral de la práctica tiene influencia sobre la nota final obtenida en la misma, de forma que puede mejorar o empeorar la nota parcial calculada a partir del cumplimiento de objetivos, las normas de estilo del código fuente y la memoria. Si el desempeño del alumno en la defensa oral no alcanza el mínimo nivel admisible, la práctica en su conjunto se considerará SUSPENSA, pudiéndose incluso solicitar al alumno que entregue una práctica totalmente nueva para la siguiente convocatoria.

Evaluación de la práctica

La nota de la práctica será individual para cada alumno, y se calculará a partir de:

- El cumplimiento de los objetivos planteados en la misma.
- La aplicación de las normas de estilo.
- La memoria.
- La defensa oral individual.

Autoría de la práctica

La práctica se debe realizar en grupos de 2 personas como máximo.

El hecho de detectar copia en las prácticas expondrá a los alumnos a la posibilidad de una apertura de expediente disciplinario y expulsión. **En caso de detectar copia, los alumnos afectados serán suspendidos en la TOTALIDAD de la asignatura.** Una práctica será considerada copia en caso de que contenga la totalidad o una parte de la práctica de otro alumno. Se considerará copia en caso de:

- Archivos que contengan la totalidad o fragmentos de código de otro alumno
- Memorias con la totalidad o fragmentos de frases e imágenes de otro alumno

El equipo docente hará uso de detectores automáticos de plagio en las prácticas, tanto en la parte referente al código como a la memoria.

Apéndice

En el Campus Virtual están colgados dos archivos necesarios para realizar esta práctica: *parser.h* y *libparser.a* (*libparser_64.a* para sistemas Linux x86_64) que contienen una función y dos tipos de datos que se explican a continuación. También está colgado el archivo *test.c* que contiene un programa de ejemplo para utilizar la librería *minishell*.

Función *tline* tokenize(char * str)*:

Recibe como argumento un puntero a una cadena de caracteres, y devuelve un puntero a una variable de tipo *tline* que contiene la información sobre la cadena analizada.

Tipo de datos *tline*:

Representa los datos extraídos de una línea de mandatos. Contiene varios campos:

int ncommands: número de mandatos que hay en la línea.

*tcommand *commands*: una array de elementos del tipo *tcommand*.

*char *redirect_input*: si la línea tiene redirección de entrada, *redirect_input* es un puntero a un string con el nombre del fichero para la redirección, si no tiene redirección de entrada *redirect_input* apunta a NULL.

*char *redirect_output*: igual que el anterior, pero referido a la redirección de salida.

*char *redirect_error*: igual que el anterior, pero referido a la redirección de error.

int background: activado a 1 si el mandato se ejecuta en background o 0 en caso contrario.

Tipo de datos *tcommand*:

Representa un mandato. Contiene 3 campos:

*char *filename*: string con la ruta completa del mandato, o NULL si el mandato no existe.

int argc y *char **argv*: similares a los argumentos de una función *main*, pero referidos al mandato.

Como compilar un programa que hace uso de la librería *parser*:

El código del programa debe incluir la directiva *#include "parser.h"*.

Para compilar el programa de ejemplo que está en el Campus Virtual, deben colocarse los archivos *test.c*, *parser.h* y la librería *libparser.a* correspondiente en un mismo directorio, e introducir el siguiente mandato:

gcc test.c libparser.a -o test

Si se utiliza cualquier editor de apoyo (Eclipse, ...) será necesario incluir la librería *parser* (*-lparser*), el directorio donde esté (*-L/directorio*) y el directorio donde se encuentre su fichero *.h* correspondiente (*-i/directorio*)