Práctica Sistemas Operativos: Llamadas al sistema para la gestión de procesos

Esta práctica permite al alumno familiarizarse con los servicios para la gestión de procesos que proporciona `POSIX`. Asimismo, se pretende que conozca cómo es el funcionamiento interno de un intérprete de mandatos en UNIX/Linux.

1 Objetivo de la práctica

El alumno deberá diseñar y codificar, en lenguaje C y sobre sistema operativo UNIX/Linux, un programa que actúe como intérprete de mandatos o *shell*. El programa deberá seguir estrictamente las especificaciones y requisitos contenidos en este documento. Con la realización de este programa el alumno adquirirá valiosos conocimientos de programación en entorno UNIX. Tanto en el uso de las llamadas al sistema operativo (`FORK`, `EXEC`, `SIGNAL`, `PIPE`, `DUP`, etc.), como en el manejo de herramientas como el visualizador de páginas de manual (`man`), el compilador de C (`gcc`), el regenerador de programas (`make`), etc.

NOTA: Durante la lectura de este documento encontrará la notación "`man # xxxxx`", que sugiere usar el mandato `man` de UNIX/Linux para obtener información sobre el comando xxxxx de la sección #. Haga caso de las recomendaciones.

2 Descripción de la práctica

El intérprete de mandatos a desarrollar o *minishell* utiliza la entrada estándar (descriptor de fichero 0), para leer las líneas de mandatos que interpreta y ejecuta. Utiliza la salida estándar (descriptor de fichero 1) para presentar el resultado de los comandos internos. Y utiliza el estándar error (descriptor de fichero 2) para notificar los errores que se puedan dar. Si ocurre un error en alguna llamada al sistema, se utiliza para notificarlo la función de librería `perror`.

Para el desarrollo de esta práctica se proporciona al alumno el *parser* que permite leer los mandatos introducidos por el usuario. El alumno sólo deberá preocuparse de implementar el intérprete de mandatos. La sintaxis que utiliza este *parser* es la siguiente:

Blanco

Es un carácter tabulador o espacio.

Separador

Es un carácter con significado especial (|, <, >, &), el fin de línea o el fin de fichero (por teclado `CTRL-D`).

Texto

Es cualquier secuencia de caracteres delimitada por blanco o separador.

Mandato

Es una secuencia de textos separados por blancos. El primer texto especifica el nombre del mandato a ejecutar. Las restantes son los argumentos del mandato invocado. El nombre del mandato se pasa como argumento 0 (`man execvp`). Cada mandato se ejecuta como un proceso hijo directo del *minishell* (`man fork`). El valor de un mandato es su estado de terminación (`man 2 wait`). Si la ejecución falla se notifica el error (por el estándar error).

Secuencia
Es una secuencia de dos o más mandatos separados por '|'. La salida estándar de cada mandato se conecta por una tubería (`man pipe`) a la entrada estándar del siguiente. El *minishell* normalmente espera la terminación del último mandato de la secuencia antes de solicitar la siguiente línea de entrada. El valor de una secuencia es el valor del último mandato de la misma.

Redirección
La entrada o la salida de un mandato o secuencia puede será redirigida añadiendo tras él la siguiente notación:

- **< fichero**
  Usa fichero como entrada estándar abriéndolo para lectura (`man open`).
- **> fichero**
  Usa fichero como salida estándar. Si el fichero no existe se crea, si existe se trunca (`man creat`), modo de creación `0666`.
- **>& fichero**
  Usa fichero como estándar error. Si el fichero no existe se crea, si existe se trunca (`man creat`), modo de creación `0666`.

En caso de cualquier error durante las redirecciones, se notifica (por la salida estándar de error) y se suspende la ejecución de la línea.

Background (&)
Un mandato o secuencia terminado en '`&`'supone la ejecución asíncrona del mismo, esto es, el *minishell* no queda bloqueado esperando su terminación. Ejecuta el mandato sin esperar por él imprimiendo por pantalla el identificador del proceso por el que habría esperado con el siguiente formato "`[%d] \n`".

prompt
Mensaje de apremio antes de leer cada línea. Por defecto será:
"`msh>`"

2.1 Obtención de la línea de mandatos leída.

Para obtener la línea de mandatos tecleada por el usuario debe utilizarse la función `obtain_order` cuyo prototipo es el siguiente:

```
int obtain_order(char ***argvv,char **filev,int *bg);
```

La llamada devuelve 0 en caso de teclear `Control-D` (`EOF`), -1 si se encontró un error. Si se ejecuta con éxito la llamada devuelve el número de mandatos más uno. Así:

- Para `ls -l` devuelve 2
- Para `ls -l | sort` devuelve 3

El argumento `argvv` permite tener acceso a todos los mandatos introducidos por el usuario.
Con el argumento `filev` se pueden obtener los ficheros utilizados en la redirección:

- `filev[0]` apuntará al nombre del fichero a utilizar en la redirección de entrada en caso de que exista o `NULL` si no hay ninguno.
- `filev[1]` apuntará al nombre del fichero a utilizar en la redirección de salida en caso de que exista o `NULL` si no hay ninguno.

- `filev[1]` apuntará al nombre del fichero a utilizar en la redirección de la salida de error en caso de que exista o `NULL` si no hay ninguno.

El argumento `bg` es 1 si el mandato o secuencia de mandatos debe ejecutarse en *background*.
Si el usuario teclea `ls -l | sort < fichero`. Los argumentos anteriores tendrán la disposición que se muestra en la siguiente figura:

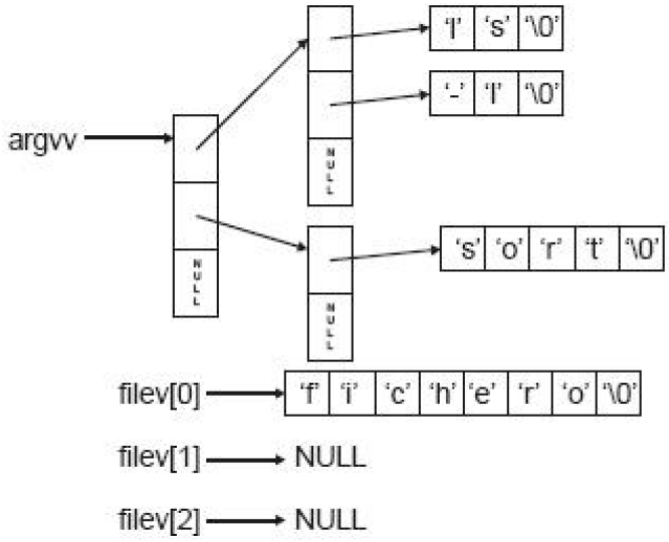

Figura 1: Estructura de datos utilizadas por parser

En el fichero `main.c` (fichero que debe rellenar el alumno con el código del *minishell*) se invoca a la función `obtain_order` y se ejecuta el siguiente bucle:

```
for (argvc = 0; (argv = argvv[argvc]); argvc++) {
      for (argc = 0; argv[argc]; argc++)
            printf(" %s ", argv[argc]);
printf("\n");
}
if (filev[0]) printf("<%s\n", filev[0]); /* IN */
if (filev[1]) printf(">%s\n", filev[1]);/* OUT */
if (filev[2]) printf(">& %s\n", filev[2]);/* ERR */
if (bg) printf("&\n");}
```

Se recomienda al alumno que, sin modificar este fichero, compile y ejecute el *minishell*, introduciendo diferentes mandatos y secuencias de mandatos para comprender claramente como acceder a cada uno de los mandatos de una secuencia.

2.2 Desarrollo del *minishell*

Para desarrollar el *minishell* se recomienda al alumno seguir una serie de pasos, de tal forma que se construya el *minishell* de forma incremental. En cada paso se añadirá nueva funcionalidad sobre el anterior.

1. Ejecución de mandatos simples del tipo `ls -l`, `who`, etc.
2. Ejecución de mandatos simples con redirecciones (entrada, salida y de error).
3. Ejecución de mandatos simples en *background*.
4. Ejecución de secuencias de mandatos conectados por *pipes*.

5. Tratamiento de señales. Ni el *minishell* ni los comandos lanzados en background, deben morir por señales generadas desde teclado (`SIGINT`, `SIGQUIT`) (`man signal`), por lo tanto éstas son ignoradas. Por contra, los comandos lanzados en *foreground* deben morir si le llegan estas señales, por lo tanto mantienen la acción por defecto.

3 Código Fuente de Apoyo

Se proporcionan los siguientes ficheros como código fuente de apoyo para facilitar la realización de esta práctica:

Makefile

Fichero fuente para la herramienta `make`. NO debe ser modificado. Con él se consigue la recompilación automática sólo de los ficheros fuente que se modifiquen (`make msh`).

y.c

Fichero fuente de C. NO debe ser modificado. Define funciones básicas para usar la herramienta `lex` sin necesidad de usar la librería `l`.

scanner.l

Fichero fuente para la herramienta `lex`. NO debe ser modificado. Con él se genera automáticamente código C que implementa un analizador lexicográfico (scanner) que permite reconocer el *token* `TXT`, considerando los posibles separadores (`\t | < > & \n`).

parser.y

Fichero fuente para la herramienta `yacc`. NO debe ser modificado. Con él se genera automáticamente código C que implementa un analizador gramatical (*parser*) que permite reconocer sentencias correctas de la gramática de entrada del *minishell*.

main.c

Fichero fuente de C que muestra como usar el *parser*. Este fichero es el que se DEBE MODIFICAR.

Se recomienda estudiar detalladamente para la correcta comprensión del uso de la función de interfaz, `obtain_order`. La versión que se ofrece hace eco de las líneas tecleadas que sean sintácticamente correctas. Esta funcionalidad debe ser eliminada y sustituida por la ejecución de las líneas tecleadas.

4 Recomendaciones generales

- Desarrolle el *minishell* por etapas, complicándolo progresivamente, tal y como se especifica en la sección Desarrollo del *minishell*.
- Para ello lea detenidamente este documento y sea estricto con la información en él contenida, en concreto con los formatos de presentación de los comandos.
- Lea asimismo las páginas de manual a las que se hace referencia. Cuando tenga una idea clara de cómo implementar lo que se le pide, codifíquelo, compile y pruebe su práctica.