

Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid



# SISTEMAS OPERATIVOS

Práctica 2. Programación de un intérprete de mandatos (Minishell)

**Grado de Ingeniería en Informática**

Curso 2016/2017

## Índice

<a href="#">1. Enunciado de la Práctica</a>	2
<a href="#">1.1. Descripción de la Práctica</a>	2
<a href="#">1.1.1. Parser proporcionado</a>	2
<a href="#">1.1.2. Obtención de la línea de mandatos.</a>	4
<a href="#">1.1.3. Desarrollo del minishell</a>	5
<a href="#">a) Mandato interno: mycalc</a>	6
<a href="#">b) Mandato interno: mybak</a>	6
<a href="#">1.2 Código Fuente de Apoyo</a>	7
<a href="#">1.3 Corrector proporcionado</a>	8
<a href="#">2. Entrega</a>	9
<a href="#">2.1. Plazo de entrega</a>	9
<a href="#">2.2. Procedimiento de entrega de las prácticas</a>	9
<a href="#">2.3 Documentación a Entregar</a>	9
<a href="#">3. Normas</a>	10
<a href="#">4. Anexos</a>	11
<a href="#">4.1 man function</a>	11
<a href="#">4.2 Modo background y foreground</a>	11
<a href="#">4.3 Mandatos internos</a>	12
<a href="#">5. Bibliografía</a>	12

## 1. Enunciado de la Práctica

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 (shell) en UNIX/Linux. En resumen, una shell permite al usuario comunicarse con el kernel del sistema operativo mediante la ejecución de comandos o mandatos, ya sean simples o encadenados.

Para la gestión de procesos pesados, se utilizarán las llamadas al sistema de POSIX relacionadas como fork, wait, exit. Para la comunicación entre procesos, igualmente se utilizarán las llamadas al sistema pipe, dup, close, signal.

El alumno debe diseñar y codificar, en lenguaje C y sobre sistema operativo UNIX/Linux, un programa que actúe como intérprete de mandatos o shell. El programa debe seguir estrictamente las especificaciones y requisitos contenidos en este documento.

### 1.1. 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 por pantalla 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.

#### 1.1.1. Parser proporcionado

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**

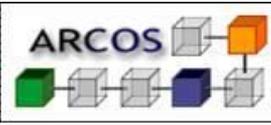
Es un carácter tabulador o espacio.

##### **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.

	<p style="text-align: center;">Departamento de Informática Grado en Ingeniería Informática Sistemas Operativos (2016-2017)</p> <p style="text-align: center;">Práctica 2 - Minishell</p>	
---	--	---

### **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 2 `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 2 `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 ser redirigida añadiendo tras él la siguiente notación:

- **< fichero** → Usa fichero como entrada estándar abriéndolo para lectura (man 2 `open`).
- **> fichero** → Usa fichero como salida estándar. Si el fichero no existe se crea, si existe se trunca (man 2 `open` / man `creat`).
- **>& fichero** → Usa fichero como estándar error. Si el fichero no existe se crea, si existe se trunca (man 2 `open` / man `creat`).

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

### **Background (&)**

Un mandato o secuencia terminado en '&' supone la ejecución asíncrona del mismo, es decir, el minishell no queda bloqueado esperando su terminación. Ejecuta el mandato sin esperar por él, imprimiendo por pantalla su *pid*. Para hacer esta impresión se debe seguir el siguiente formato, donde %d indica el *pid* del proceso por el que no se espera:

[%d]\n

### **Prompt**

Mensaje de espera por el usuario, antes de leer cada línea. Por defecto será:

**"msh>"**

	<p style="text-align: center;">Departamento de Informática Grado en Ingeniería Informática Sistemas Operativos (2016-2017)</p> <p style="text-align: center;">Práctica 2 - Minishell</p>	
---	--	---

### 1.1.2. Obtención de la línea de mandatos.

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), y devuelve -1 si se encontró un error. **Si se ejecuta con éxito la llamada, devuelve el número de mandatos +1.** 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 apunta a NULL si no hay ninguno.
- **filev[1]** apunta al nombre del fichero a utilizar en la redirección de salida en caso de que exista o apunta a NULL si no hay ninguno.
- **filev[2]** apunta al nombre del fichero a utilizar en la redirección de la salida de error en caso de que exista o apunta a NULL si no hay ninguno.

El argumento *bg* es 1 si el mandato o secuencia de mandatos debe ejecutarse en background.

**EJEMPLO:** Si el usuario teclea

```
ls -l | sort < fichero
```

los argumentos anteriores tendrán la disposición que se muestra en la siguiente figura:

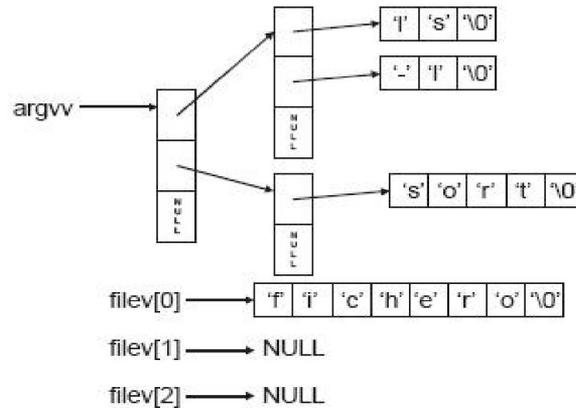


Figura 1: Estructura de datos generada por el *parser*.

En el fichero `msh.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 (command_counter = 0; command_counter < num_commands;
command_counter++)
{
    for (args_counter = 0; (argvv[command_counter][args_counter] != NULL);
args_counter++)
    {
        printf("%s ", argvv[command_counter][args_counter]);
    }
    printf("\n");
}
if (filev[0] != NULL) printf("<%s\n", filev[0]); // IN
if (filev[1] != NULL) printf(">%s\n", filev[1]); // OUT
if (filev[2] != NULL) printf(">& %s\n", filev[2]); // ERR
if (bg) printf("&\n");

```

Se recomienda al alumno que antes de comenzar a implementar la práctica, compile y ejecute el minishell, introduciendo diferentes mandatos y secuencias de mandatos para comprender claramente cómo acceder a cada uno de los mandatos de una secuencia.

### 1.1.3. 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 en background.
3. Ejecución de secuencias de mandatos conectados por pipes. El número de mandatos, en una secuencia, se limitará a 3, es decir: `ls -l | sort | wc`. Si los alumnos deciden implementar una secuencia indeterminada de pipes (no limitada a 3, ni a ninguna cantidad), se considerará para nota complementaria.
4. Ejecución de mandatos simples y secuencias de mandatos con redirecciones (entrada, salida y de error) y en background.
5. Ejecución de mandatos internos. Un mandato interno es aquel que bien se corresponde directamente con una llamada al sistema o bien es un complemento que ofrece el propio minishell. Para que su efecto sea permanente, ha de ser implementado y ejecutado dentro del propio minishell (en el proceso padre). Todo mandato interno comprueba el número de argumentos con que se le invoca y si encuentra este o cualquier otro error, lo notifica (por el estándar error) y termina con valor distinto de cero. Los mandatos internos que se piden en el minishell son:

a) Mandato interno: **mycalc**

Funciona como una calculadora muy sencilla en el terminal. Toma una ecuación simple, con la forma *operando operador operando*, donde *operando* es un número entero y *operador* puede ser la suma (`add`) o el módulo (`mod`), en la que se deberá calcular el cociente entero y el resto de la división.

Para la suma, adicionalmente se almacenarán los valores obtenidos en una variable de entorno llamada “Acc”. Dicha variable comienza valiendo 0, y posteriormente se van sumando los resultados de las sumas, pero no de los módulos. Para el módulo, se mostrará el resultado en forma de **Dividendo = Divisor \* Cociente + Resto**

Si la operación tiene éxito, se muestra (*por el estándar error*) el resultado de resolver el cálculo precedido de la etiqueta [OK]. En el caso de la suma, además se deberá mostrar el valor acumulado.

Si el operador no se correspondiese con los dos establecidos, o no se introdujesen todos los términos de la ecuación, se mostrará (por la salida estándar) el mensaje “[ERROR] La estructura del comando es `<operando 1> <add/mod> <operando 2>`”

A continuación se muestran algunos ejemplos de uso:

```
msh> mycalc 3 add -7
[OK] 3 + -7 = -5; Acc -4
```

```
msh> mycalc 5 add 12
[OK] 5 + 12 = 17; Acc 13
msh> mycalc 10 mod 7
[OK] 10 % 7 = 7 * 1 + 3
msh> mycalc 10 % 7
[ERROR] La estructura del comando es <operando 1> <add/mod> <operando 2>
msh> mycalc 8 mas
[ERROR] La estructura del comando es <operando 1> <add/mod> <operando 2>
```

### b) Mandato interno: **mybak**

Funciona de manera similar al comando 'cp'. Toma un fichero y un directorio, e intenta copiar el fichero elegido en el directorio especificado, poniéndole el mismo nombre que el del archivo original con permisos de escritura y lectura tanto para el propietario como para su grupo.

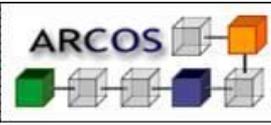
Si no se pueden abrir alguno de los dos ficheros, se deberá mostrar (*por la salida standard*) el mensaje “[**ERROR**] Error al abrir el fichero origen” o “[**ERROR**] Error al abrir el fichero destino”, según corresponda. Si el fichero de destino ya existe, se truncará su contenido a 0 y se escribirá en él.

Si no se llama a la función con los dos parámetros, se debe mostrar (por la salida estándar) el siguiente mensaje para mostrar el ejemplo de ejecución del mandato: “[**ERROR**] La estructura del comando es **mybak <fichero origen> <directorio destino>**”.

Si se consigue realizar la copia con éxito se debe mostrar (por la salida estándar) el siguiente mensaje: “[**OK**] Copiado con éxito el fichero <origen> a la carpeta <destino>”.

A continuación se muestran algunos ejemplos de uso:

```
msh> mybak msh.c /home/user/Escritorio
[OK] Copiado con éxito el fichero msh.c a la carpeta /home/user/Escritorio
msh> mybak inexistente.c .
[ERROR] Error al abrir el fichero origen
: No such file or directory
msh> mybak origen.txt
[ERROR] La estructura del comando es mybak <fichero origen> <directorio destino>
```

	<p style="text-align: center;">Departamento de Informática Grado en Ingeniería Informática Sistemas Operativos (2016-2017)</p> <p style="text-align: center;">Práctica 2 - Minishell</p>	
---	--	---

## 1.2 Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero *p2\_minishell\_2017.tgz* que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

```
tar zxvf p2_minishell_2017.tgz
```

Al extraer su contenido, se crea el directorio *ssoo\_p2\_msh/*, donde se debe desarrollar la práctica. Dentro de este directorio se habrán incluido los siguientes ficheros:

### **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.

### **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 (nt j < > & nn).

### **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.

### **msh.c**

Fichero fuente de C que muestra cómo usar el parser. **Este fichero es el que se DEBE MODIFICAR PARA HACER LA PRÁCTICA**. Se recomienda estudiar detalladamente para la correcta comprensión de la práctica, la función *obtain\_order*. La versión actual que se ofrece hace 'eco' (*eco = impresión por pantalla*) de las líneas tecleadas que sean sintácticamente correctas. Esta funcionalidad debe ser eliminada y sustituida por las líneas de código que implementan la práctica.

**NOTA 1:** Para la compilación de la práctica es necesario tener instalado los paquetes correspondientes a **Yacc** y **Lex**.

En caso de desarrollar la práctica fuera de las aulas informáticas en ordenadores personales es necesario tener los paquetes correspondientes al analizador léxico y sintáctico Yacc y Lex.

En el caso de sistemas Ubuntu / Debian, basta con instalar los paquetes 'byacc' y 'flex' de la siguiente manera.

**sudo apt-get install byacc flex**

En caso de tener otro sistema operativo, será necesario buscar el paquete equivalente para cada distribución.

## 2. Entrega

### 2.1. Plazo de entrega

La fecha límite de entrega de la práctica en AULA GLOBAL será el **martes 28 de Marzo de 2017 (hasta las 23:55h)**.

### 2.2. Procedimiento de entrega de las prácticas

La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas. En concreto, **se habilitará un entregador para el código de la práctica y otro de tipo TURNITIN para la memoria de la práctica.**

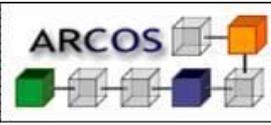
### 2.3 Documentación a Entregar

En el entregador para el código se debe entregar un archivo comprimido en formato zip con el nombre `ssoo_p2_AAAAAAAAAA_BBBBBBBBBB.zip` donde A...A y B...B son los NIAs de los integrantes del grupo. El archivo debe contener:

- `msh.c`

En el entregador TURNITIN deberá entregarse el fichero `memoria.pdf`. La memoria tendrá que contener al menos los siguientes apartados:

- **Portada:** con los nombres completos de los autores, NIAs y direcciones de correo electrónico.
- **Índice**
- **Descripción del código** detallando las principales funciones implementadas. NO incluir código fuente de la práctica en este apartado. Cualquier código será automáticamente ignorado.
- **Batería de pruebas** utilizadas y resultados obtenidos. Se dará mayor puntuación a

	<p style="text-align: center;">Departamento de Informática Grado en Ingeniería Informática Sistemas Operativos (2016-2017)</p> <p style="text-align: center;">Práctica 2 - Minishell</p>	
---	--	---

pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:

- Que un programa compile correctamente y sin advertencias (*warnings*) no es garantía de que funcione correctamente.
- Evite pruebas duplicadas que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.

No se admiten captura de pantalla.

- **Conclusiones**, problemas encontrados, cómo se han solucionado, y opiniones personales.

Se puntuará también los siguientes aspectos relativos a la **presentación** de la práctica:

- Debe contener portada, con los autores de la práctica y sus NIAs.
- Debe contener índice de contenidos.
- La memoria debe tener números de página en todas las páginas (menos la portada).
- El texto de la memoria debe estar justificado.

**La longitud de la memoria no deberá superar las 15 páginas** (portada e índice incluidos). Es imprescindible aprobar la memoria para aprobar la práctica, por lo que no debe descuidar la calidad de la misma.

***NOTA:*** La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

### 3. Normas

- 1) **Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.**
- 2) **Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, los alumnos involucrados (copiados y copiadores) perderán las calificaciones obtenidas por evaluación continua.**
- 3) **Los programas deben compilar sin warnings.**
- 4) **Los programas deberán funcionar bajo un sistema Linux, no se permite la realización de la práctica para sistemas Windows. Además, para asegurarse del correcto funcionamiento de la práctica, deberá chequearse su compilación y ejecución en los laboratorios de informática de la universidad o en el servidor [guernika.lab.inf.uc3m.es](http://guernika.lab.inf.uc3m.es). Si el código presentado no compila o no funciona sobre estas plataformas la implementación no se considerará correcta.**
- 5) **Un programa no comentado, obtendrá una calificación de 0.**
- 6) **La entrega de la práctica se realizará a través de aula global, tal y como se detalla en el apartado Entrega de este documento. No se permite la entrega a través de correo electrónico sin autorización previa.**
- 7) **Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.**
- 8) **Se debe realizar un control de errores en cada uno de los programas.**

**Los programas entregados que no sigan estas normas no se considerarán aprobados.**

## 4. Anexos

### 4.1 *man function*

**man** es el paginador del manual del sistema, es decir permite buscar información sobre un programa, una utilidad o una función. Véase el siguiente ejemplo:

#### *man 2 fork*

Las páginas usadas como argumentos al ejecutar *man* suelen ser normalmente nombres de programas, utilidades o funciones. Normalmente, la búsqueda se lleva a cabo en todas las secciones de manual disponibles según un orden predeterminado, y sólo se presenta la primera página encontrada, incluso si esa página se encuentra en varias secciones.

Para salir de la página mostrada, basta con pulsar la tecla 'q'.

Una página de manual tiene varias partes. Éstas están etiquetadas como NOMBRE, SINOPSIS, DESCRIPCIÓN, OPCIONES, FICHEROS, VÉASE TAMBIÉN, BUGS, y AUTOR. En la etiqueta de SINOPSIS se recogen las librerías (identificadas por la directiva *#include*) que se deben incluir en el programa en C del usuario para poder hacer uso de las funciones correspondientes.

Las formas más comunes de usar *man* son las siguientes:

- **man sección elemento:** Presenta la página de elemento disponible en la sección del manual.
- **man -a elemento:** Presenta, secuencialmente, todas las páginas de elemento disponibles en el manual. Entre página y página se puede decidir saltar a la siguiente o salir del paginador completamente.
- **man -k palabra-clave:** Busca la palabra-clave entre las descripciones breves y las páginas de manual y presenta todas las que casen.

### 4.2 *Modo background y foreground*

Cuando una secuencia de mandatos se ejecuta en background, el *pid* que se imprime es el del proceso que ejecuta el último mandato de la secuencia.

Cuando un mandato simple se ejecuta en background, el *pid* que se imprime es el del proceso que ejecuta ese mandato.

Con la operación de background, es posible que el proceso minishell muestre el prompt entremezclado con la salida del proceso hijo. Esto es correcto.

	<p style="text-align: center;"> <b>Departamento de Informática</b>  <b>Grado en Ingeniería Informática</b>  <b>Sistemas Operativos (2016-2017)</b>  <b>Práctica 2 - Minishell</b> </p>	
---	--	---

Después de ejecutar un mandato en foreground, la minishell no puede tener procesos zombies de mandatos anteriores ejecutados en background.

### **4.3 Mandatos internos**

Los mandatos internos (mycalc y mybak) se ejecutan en el proceso minishell, y por tanto:

- No forman parte de secuencias de mandatos
- No tienen redirecciones de ficheros
- No se ejecutan en background

## **5. Bibliografía**

- El lenguaje de programación C: diseño e implementación de programas Félix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- The UNIX System S.R. Bourne Addison-Wesley, 1983.
- Advanced UNIX Programming M.J. Rochkind Prentice-Hall, 1985.
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Programming Utilities and Libraries SUN Microsystems, 1990.
- Unix man pages (man function)