

Área de Arquitectura y Tecnología de Computadores

Universidad Carlos III de Madrid



# SISTEMAS OPERATIVOS

Práctica 1. Llamadas al sistema

**Grado de Ingeniería en Informática**

**Doble Grado en Ingeniería en Informática y ADE**

Curso 2017/2018

## Índice

<b>Introducción</b>	<b>3</b>
<b>Descripción de la Práctica</b>	<b>4</b>
printFile	4
statistics	4
split	5
filter	5
combine	5
<b>Código Fuente de Apoyo</b>	<b>7</b>
<b>Entrega</b>	<b>8</b>
Plazo de entrega	8
Procedimiento de entrega de las prácticas	8
Documentación a Entregar	8
<b>Normas</b>	<b>10</b>
<b>Anexo I: llamadas al sistema.</b>	<b>11</b>
Llamadas al sistema relacionadas con ficheros	11
<b>Anexo II: función man.</b>	<b>14</b>
<b>Bibliografía</b>	<b>15</b>



# 1. Introducción

Esta práctica permite al alumno familiarizarse con las llamadas al sistema de ficheros siguiendo el estándar POSIX. Unix permite efectuar llamadas al sistema directamente desde un programa realizado en un lenguaje de alto nivel, en particular en lenguaje C.

La mayor parte de las entradas/salidas (E/S) en Unix pueden realizarse utilizando llamadas como `open`, `read`, `write`, `lseek`, `stat` y `close`.

Para el núcleo del sistema operativo, todos los ficheros abiertos son identificados por medio de *descriptores de fichero*. Un descriptor de fichero es un entero no negativo. Cuando abrimos (`open`) un fichero que ya existe o creamos (`creat`) un nuevo fichero, el sistema operativo devuelve un descriptor de fichero al proceso. Cuando queremos leer o escribir de/en un fichero identificamos el fichero con el descriptor de fichero que fue devuelto por las llamadas anteriormente descritas.

Cada fichero abierto tiene una *posición de lectura/escritura actual* (`current_offset`). Está representado por un entero no negativo que mide el número de bytes desde el comienzo del fichero. Las operaciones de lectura y escritura comienzan normalmente en la posición actual y provocan un incremento en dicha posición, igual al número de bytes correctamente leídos o escritos. Por defecto, esta posición es inicializada a 0 cuando se abre un fichero, a menos que se especifique la opción `O_APPEND`. La posición actual (`current_offset`) de un fichero abierto puede cambiarse explícitamente utilizando la llamada al sistema `lseek`.

A la hora de obtener las propiedades de un fichero o directorio (los *metadatos* o *atributos*), se utilizan las llamadas al sistema de la familia `stat`. Algunos de esos atributos son: tipo de protección, tamaño total en bytes, fecha de último acceso, etc.



## 2. Descripción de la Práctica

Se pretende implementar una serie de funciones que permitan el manejo de datos de personas, recogidos en ficheros binarios. El fichero binario está compuesto por estructuras, cada una de las cuáles representa a una persona con los siguientes campos (se indica también el tamaño de ocupan):

- Nombre [48 bytes]: cadena de caracteres.
- Edad [4 bytes]: valor entero sin signo comprendido entre 0 y 150 (inclusive).
- DNI [4 bytes]: valor entero sin signo comprendido entre 0 y 99999999 (inclusive).
- Caracter de control del DNI [1 byte]: caracter.
- Salario [8 bytes]: valor decimal comprendido entre 0.0 y 100000.0 (inclusive).

Los distintos programas a codificar se especifican a continuación.

NOTA: las operaciones de lectura/escritura de ficheros se harán usando la estructura de declarada anteriormente (y no campos individuales).

### printFile

Programa que muestra por pantalla el contenido de un fichero de personas.

Se debe mostrar cada persona en una línea, y los atributos de una persona separados por tabulador ('\t'). El atributo salario se debe mostrar como número entero redondeado. A continuación, se muestra un ejemplo de la salida:

<i>Nombre1</i>	32	09834320	A	24500
<i>Nombre2</i>	35	32478973	V	27456
<i>Nombre3</i>	53	98435834	P	97000

Uso: `./printFile <fichero de entrada>`

### statistics

Programa que muestra las estadísticas por pantalla de un fichero de personas. Los datos a mostrar son:

- Renta media.
- Edad media.

- Caracter de control de DNI más frecuente. En caso de empate, mostrar el primero en orden alfabético.
- Edad media de las personas con el caracter de control de DNI más frecuente.

Mostrar cada valor estadístico (como valor entero redondeado, no como número real) en una línea, de la siguiente manera:

```
Renta media: 34750
Edad media: 46
Caracter de control de DNI más frecuente: S
Edad media para el caracter de control de DNI más frecuente: 32
```

Uso: `./statistics <fichero de entrada>`

## split

Programa que divide las personas del fichero de entrada según su rango de edad. El programa recibirá como parámetro por teclado un valor numérico entero positivo que define el límite del rango. Como resultado se tendrán por tanto dos ficheros de salida: uno con las personas menores que el valor indicado, y otro con las mayores o iguales.

Si los ficheros de salida ya existen, se debe añadir la nueva información al final del fichero. Si los ficheros de salida no existen, se crearán.

Uso: `./split <edad límite> <fichero de entrada> <fichero de salida para menores> <fichero de salida para mayores o iguales>`

## filter

Programa que filtra las personas con un determinado caracter de control de DNI. Como resultado se tendrá un fichero de salida con las personas cuyo caracter de control de DNI es igual al especificado como parámetro.

Si el fichero de salida ya existe, se debe truncar. Si el fichero de salida no existe, se creará.

Uso: `./filter <caracter de control> <fichero de entrada> <fichero de salida>`



## combine

Programa que combina datos de dos ficheros. Como resultado se tendrá un fichero de salida con todos las personas del fichero 1 y todas las personas del fichero 2. El orden de combinación de ambos ficheros que se debe seguir es alternando personas de uno y otro fichero (comenzando por el fichero 1 de entrada) y desde el final hacia el comienzo de los ficheros. A continuación se muestran algunos ejemplos:

### Ejemplo:

**Fichero 1 de entrada: A B C D**

**Fichero 2 de entrada: E F G H I J**

**Fichero salida: D J C I B H A G F E**

### Ejemplo:

**Fichero 1 de entrada: A B C D**

**Fichero 2 de entrada: E F**

**Fichero salida: D F C E B A**

Si el fichero de salida ya existe, se debe truncar. Si el fichero de salida no existe, se creará.

Uso: `./combine <fichero 1> <fichero 2> <fichero de salida>`

Nota: Para el acceso directo a una posición del fichero utilizar `lseek` y para conocer el tamaño del mismo utilizar `stat` (*man 2 stat*).



### 3. Código Fuente de Apoyo

Para facilitar la realización de esta práctica se dispone del fichero *llamadas\_1718.tar.xz* que contiene código fuente de apoyo. Para extraer su contenido ejecutar lo siguiente:

```
tar xvf llamadas_1718.tar.xz
```

Al extraer su contenido, se crea el directorio *ssoo/p1\_llamadas/*, 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.

#### **persona.h**

Fichero de cabecera que contiene la definición de una persona con sus atributos. **NO DEBE SER MODIFICADO.**

#### **printFile.c**

Fichero fuente de C que debe contener el programa **printFile**. **DEBE SER RELLENADO POR EL ALUMNO.**

#### **statistics.c**

Fichero fuente de C que debe contener el programa **statistics**. **DEBE SER RELLENADO POR EL ALUMNO.**

#### **split.c**

Fichero fuente de C que debe contener el programa **split**. **DEBE SER RELLENADO POR EL ALUMNO.**

#### **filter.c**

Fichero fuente de C que debe contener el programa **filter**. **DEBE SER RELLENADO POR EL ALUMNO.**

#### **combine.c**

Fichero fuente de C que debe contener el programa **combine**. **DEBE SER RELLENADO POR EL ALUMNO.**

#### **samples/**

Este directorio contiene un fichero de personas de ejemplo, para poder ejecutar la práctica.



## 4. Entrega

### 4.1. Plazo de entrega

La fecha límite de entrega de la práctica en AULA GLOBAL será el **Viernes 2 de marzo de 2018 a las 23:55.**

### 4.2. 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. **NO SE ADMITEN ENTREGAS POR CORREO ELECTRÓNICO.**

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

### 4.3. Documentación a entregar



Se debe entregar un fichero comprimido en formato zip con el nombre `ssoo_p1_AAAAAAAAAA_BBBBBBBBBB_CCCCCCCC.zip` donde A...A, B...B and C...C son los NIA's de los integrantes del grupo. El fichero debe contener:

- **memoria.pdf**
- **printFile.c**
- **persona.h**
- **statistics.c**
- **split.c**
- **filter.c**
- **combine.c**

La memoria tendrá que contener al menos los siguientes apartados:

- Índice de contenidos.
- Autores (incluyendo nombre completo, NIA y dirección de correo electrónico).



 <p>Universidad Carlos III de Madrid</p>	<p>Departamento de Informática Sistemas Operativos (2017-2018)</p> <p>Práctica 1 - Llamadas al sistema de ficheros</p>	
---	--	---

- Descripción del diseño del código, detallando las principales funciones implementadas. **NO INCLUIR CÓDIGO EN LA MEMORIA.**
- Batería de pruebas planificada y resultados esperados.
- Conclusiones, problemas encontrados y cómo se han solucionado.

**La longitud de la memoria no deberá superar las 12 páginas** (portada e índice incluidos). Se recomienda evitar pruebas duplicadas que tengan como objetivo evaluar partes de código con similares parámetros de entrada. El plan de pruebas se evaluará dependiendo de su alcance, no del número de pruebas. Es imprescindible aprobar la memoria para aprobar la práctica, por lo que no debe descuidar la calidad de la misma.



## 5. Normas

- 1) Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.
- 2) La práctica podrá realizarse en grupos de 3 personas máximo.
- 3) La práctica debe funcionar en los ordenadores de las aulas Linux o en el servidor *guernika.lab.inf.uc3m.es*. Se puede encontrar un tutorial para acceder al servidor en el siguiente enlace:

[http://www.lab.inf.uc3m.es/wp-content/docs/manual\\_conexionSSH\\_V1.0.pdf](http://www.lab.inf.uc3m.es/wp-content/docs/manual_conexionSSH_V1.0.pdf)

- 4) Todos los ejercicios deben seguir el formato de salida que se pide en cada enunciado.
- 5) Las prácticas enviadas deben contener trabajo original. En caso de que encontrar implementaciones comunes en dos prácticas, los miembros de los grupos implicados obtendrán una calificación de 0 en la práctica, suspenderán la evaluación continua, además de que se aplicarán los procedimientos administrativos correspondientes a mala conducta.
- 6) Los programas deben compilar sin warnings. De lo contrario, se penalizará la nota de la práctica.
- 7) Que la práctica compile sin errores y sin avisos no garantiza que la implementación cumpla con los requisitos funcionales. Se recomienda probar y depurar el código para asegurar su correcto funcionamiento.
- 8) Un programa que no incluya comentarios obtendrá una calificación de 0.
- 9) 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.
- 10) Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.
- 11) Se debe realizar un control de errores en cada uno de los programas.
- 12) Se deben usar técnicas de programación estructurada. Por lo tanto deberán seguirse siempre las siguientes reglas:
  - Todos los bloques tienen un único punto de entrada al comienzo de los mismos.
  - Todos los bloques tienen un único punto de salida al final de los mismos.
  - Se deberá evitar el uso de las sentencias *goto*, *break*, *continue* y *exit* en el cuerpo de los bucles.



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

## 6. Anexo I: Llamadas al Sistema.

Las llamadas al sistema proporcionan la interfaz entre el sistema operativo y un programa en ejecución. UNIX permite efectuar llamadas al sistema directamente desde un programa realizado en un lenguaje de alto nivel, en particular en lenguaje C, en cuyo caso las llamadas se asemejan a llamadas a funciones, tal y como si estuvieran definidas en una biblioteca estándar. El formato general de una llamada al sistema es:

```
status = funcion_estandar (arg1, arg2,.....)
```

En caso de realizar una llamada sin éxito, devolvería en la variable `status` un valor -1. En la variable global `errno` se coloca el número de error, con el cual podemos obtener la asociación del error con lo que realmente ha ocurrido en el fichero `errno.h`, (contenido en la ruta: `/usr/src`. En linux: `/usr/src/linux/include/asm/errno.h`).

### 6.1. Llamadas al sistema relacionadas con ficheros

```
fd = creat(nombre_fichero, permisos)
```

Crea un nuevo fichero (vacío) dado un nombre de ruta y lo abre para la escritura sin importar el modo del fichero. Devuelve el descriptor del fichero, `fd`, el cual se puede utilizar para escribir el fichero. Si se hace `creat` en un fichero ya existente, ese fichero se trunca a la longitud 0, siempre y cuando todos los permisos sean los correctos.

Derechos: **r** - Lectura. **w** - Escritura. **x** - Ejecución.

Ejemplo: permisos = 7 1 0 ( octal )  $\rightarrow$  binario  $\rightarrow$  **r w x r w x r w x**  $\rightarrow$  1 1 1 0 0 1 0 0 0

Más ayuda en: `man 2 creat`

```
fd = open(nombre_fichero, flags, modo)
```

Abre un fichero existente. También puede utilizarse esta llamada para crear un fichero cuando no existe. Devuelve un descriptor de fichero que se puede utilizar para la lectura o escritura en el fichero.



Más ayuda en: `man 2 open`

```
n = close (fd)
```

Cierra un fichero abierto anteriormente, lo cual hace disponible el descriptor de fichero para su uso en otro `creat` u `open`. Si `n = -1` → Error al cerrar el fichero.

Más ayuda en: `man 2 close`

```
n = read(fd, direccion_mem, n°_bytes)
```

Lee de un fichero (cuyo descriptor de fichero se obtuvo de abrirlo) tantos bytes como indica `n°_bytes`, colocando la información leída a partir de la dirección de memoria `direccion_mem`.

Devuelve en `n` el número de bytes que realmente se han leído, debiendo coincidir con `n°_bytes`. Si `n = 0` → Fin de fichero (EOF). Si `n = -1` → Error de lectura.

Más ayuda en: `man 2 read`

```
n = write(fd, direccion_mem, n°_bytes)
```

Escribe en un fichero (cuyo descriptor de fichero `fd` se obtuvo al abrirlo) tantos bytes como indica `n°_bytes`, tomándolos de la dirección de memoria indicada `direccion_mem`. Devuelve en `n` el número de bytes que realmente se han escrito, debiendo coincidir con `n°_bytes`. Si `n = -1` → Error de escritura.

Cada `write` (así como cada `read`), actualiza automáticamente la posición actual del fichero que se usa para determinar la posición en el fichero del siguiente `write` o `read`.

Más ayuda en: `man 2 write`

```
x = lseek(fd, desplazamiento, origen)
```

Modifica el valor del apuntador de desplazamiento en el fichero, a la posición explícita en `desplazamiento` a partir de la referencia impuesta en `origen`, de forma que las llamadas `read` o `write` pueden iniciar en cualquier parte del fichero.

Si `x = -1` → Error de posicionamiento.

El parámetro *origen* puede tomar los siguientes valores:

- `SEEK_SET` → desde el principio del fichero.

- SEEK\_CUR → desde la posición actual.
- SEEK\_END → desde el final del fichero.

El parámetro *desplazamiento* se expresa en bytes, y toma un valor positivo o negativo.

Ejemplo:

a b c d e f g h i

`lseek(5, 4, SEEK_SET)` → Al avanzar 4 bytes, la siguiente lectura sería la "e". El descriptor del fichero abierto 'fd' es 5.

Más ayuda en: `man 2 lseek`



## 7. Anexo II: Función man.

**man** permite buscar información sobre un programa, una utilidad o una función. Véase el siguiente ejemplo:

```
man 2 open
```

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 indicada entre las descripciones breves y las páginas de manual y presenta todas las páginas resultantes.



## 8. 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`)