

**Área de Arquitectura y Tecnología de Computadores (ARCOS)**

Universidad Carlos III de Madrid

**SISTEMAS OPERATIVOS****Práctica 1. Llamadas al sistema operativo****Grado de Ingeniería en Informática****Grado en Matemática Aplicada y Computación****Doble Grado en Ingeniería Informática y Administración de Empresas**

Curso 2024/2025

## Tabla de contenido

<b>Área de Arquitectura y Tecnología de Computadores (ARCOS)</b> .....	<b>1</b>
<b>1 Enunciado de la Práctica</b> .....	<b>3</b>
<b>1.1 Descripción de la Práctica</b> .....	<b>3</b>
1.1.1    crear .....	3
1.1.2    combine.....	4
1.1.3    Código Fuente de apoyo.....	5
1.1.4    Corrector proporcionado .....	6
<b>2 Entrega</b> .....	<b>6</b>
<b>2.1 Plazo de Entrega</b> .....	<b>6</b>
<b>2.2 Procedimiento de entrega de las prácticas</b> .....	<b>6</b>
<b>2.3 Documentación a Entregar</b> .....	<b>6</b>
<b>3 Normas</b> .....	<b>8</b>
<b>4 Anexo (Llamadas al Sistema)</b> .....	<b>9</b>
<b>4.1 Llamadas al sistema relacionadas con archivos</b> .....	<b>9</b>
<b>4.2 Manual (man function)</b> .....	<b>10</b>
<b>5 Bibliografía</b> .....	<b>11</b>

# 1 Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse con las llamadas al sistema operativo (en concreto, el 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) sobre ficheros en Unix pueden realizarse utilizando solamente pocas llamadas: open, creat, read, write, lseek y close.

Para el kernel del sistema operativo, todos los archivos abiertos son identificados por medio de descriptores de fichero. Un descriptor de fichero es un entero no negativo. Cuando abrimos, open, un archivo que ya existe, el kernel devuelve un descriptor de fichero al proceso. Cuando queremos leer o escribir de/en un archivo, identificamos el archivo con el descriptor de fichero que fue devuelto por la llamada anteriormente descrita.

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

## 1.1 Descripción de la Práctica

Se pretende implementar dos programas en C que utilicen las llamadas al sistema anteriormente descritas. Dichos programas serán **crear** y **combine**. Para ello, dispondrán de los correspondientes ficheros de código *crear.c*, *combine.c*.

### 1.1.1 crear

El primer programa, **crear**, recibirá como argumentos un nombre de archivo y un modo de creación específico (en octal) y creará dicho archivo con el modo indicado. En caso de error, debe mostrar un aviso por la salida de error y devolver -1.

Obsérvese que el sistema operativo puede tener definida una máscara de creación por defecto, por lo que antes de crear el archivo hay que quitar dicha máscara y almacenarla en una variable. Tras crear el fichero hay que restaurar la máscara anterior.

**Ejemplo de uso:** `./crear <fichero> <modo>`

- **Requisitos:**

- El programa debe devolver -1 si el número de argumentos de entrada es erróneo.
  - El programa debe devolver -1 si hubo un error al crear el fichero.
  - El programa debe devolver 0 si todo funcionó correctamente.
- **Sugerencia de prueba:** Constatar que al ejecutar “ls -l” en el directorio donde se ha ejecutado el programa se ve el fichero deseado con los permisos pedidos en el modo rwxrwxrwx.

### 1.1.2 combine

El segundo programa, **combine**, combinará los datos de dos ficheros de cursos de alumnos, cuyos nombres son argumentos de entrada, almacenando la combinación en un tercer fichero de curso nuevo pasado también como argumento. Este fichero de salida deberá estar ordenado por nota de menor a mayor. **NOTA:** este fichero resultante no está escrito en modo texto. Los datos se escriben en modo binario.

La estructura de la información del alumno es la siguiente, y no puede haber dos alumnos con mismo nombre y convocatoria.

```
struct alumno{  
    char nombre[50];  
    int nota;  
    int convocatoria;  
};
```

No habrá estudiantes repetidos en los ficheros. Además, el número máximo de estudiantes permitidos es 100. Si se detectan más de 100, imprime un error y finaliza la ejecución.

Además, el programa debe clasificar a los alumnos según su nota, generando un nuevo fichero de salida de estadísticas “estadisticas.csv” (valores separados por “;”) como resultado con la clasificación de los alumnos. La clasificación a extraer es la siguiente:

- M: si tiene como nota 10.
- S: si tiene como nota 9.
- N: si tiene como nota 8 o 7.
- A: si tiene como nota 6 o 5.
- F: si tiene como nota menos de 5.

Se debe imprimir el número total de alumnos en cada categoría y el porcentaje sobre el total (con 2 decimales). Ejemplos:

M;54;23.50%

S;32;15.55%

**Uso:**

```
./combine <fichero curso 1> <fichero curso 2> <fichero curso de salida>
```

- **Requisitos:**

- El programa debe devolver **-1** si hubo un error al abrir los ficheros o al crear los de salida (e.g. el directorio no existe).
  - Si es correcto, debe devolver 0.
- **Sugerencia de prueba:** Constatar que la distribución de estadísticas es correcta y totaliza el número de alumnos mezclados.

#### 1.1.3 Código Fuente de apoyo

Para facilitar la realización de esta práctica se dispone del fichero **p1-llamadas-2025.zip** que contiene código fuente de apoyo. Para extraer su contenido ejecutar el siguiente comando:

```
$ unzip p1-llamadas-2025.zip
```

Al extraer su contenido, se crea el directorio **p1-llamadas/**, donde se debe desarrollar la práctica. Dentro de este directorio se habrán incluido los siguientes ficheros:

- Makefile  
**NO debe ser modificado.** Fichero fuente para la herramienta make. Con él se consigue la recompilación automática sólo de los ficheros fuente que se modifiquen. Utilice *make* para compilar los programas, y *make clean* para eliminar los archivos compilados.
- crear.c  
**Debe modificarse.** Fichero fuente de C donde los alumnos deberán codificar el programa **crear**.
- combine.c  
**Debe modificarse.** Fichero fuente de C donde los alumnos deberán codificar el programa **combine**.
- autores.txt  
**Debe modificarse.** Fichero en formato txt donde incluir los autores de la práctica (unompor línea). Formato:

*NIA, Apellidos, Nombre*

#### 1.1.4 Corrector proporcionado

Se proporciona a los alumnos el script en **python (versión 3) probador-ssoo-p1.py** que verifica que el formato del entregable de la práctica es el correcto (sigue las especificaciones de nombrado, y está bien comprimido) y ejecuta tests de funcionalidad, dando como resultado una nota tentativa del código proporcionado. El probador deberá ejecutarse en las máquinas virtuales con Ubuntu Linux y en la Aulas Virtuales proporcionadas por el laboratorio del Departamento de Informática. Para su correcto uso, el contenido del ZIP que contiene el probador debe ser extraído en la carpeta del proyecto. El comando para ejecutar el corrector es el siguiente:

```
$ python probador-ssoo-p1.py <entregable.zip>
```

Siendo entregable.zip el fichero que se va a entregar por Aula Global (ver siguiente apartado). Ejemplo:

```
$ python probador-ssoo-p1.py ssoo p1-100254896-100047014.zip
```

El corrector imprimirá mensajes por pantalla indicando si el formato es o no correcto.

## 2 Entrega

### 2.1 Plazo de Entrega

La fecha límite de entrega de la práctica en AULA GLOBAL será el **3 de marzo de 2025 (hasta las 23:55h)**

### 2.2 Procedimiento de entrega de las prácticas

La entrega de las prácticas ha de realizarse de forma electrónica y por **un único integrante del grupo**. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales se 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

Se debe entregar un archivo comprimido en formato zip con el nombre:

**Ssoo\_p1\_NIA1\_NIA2\_NIA3.zip**

Con los NIAs de los integrantes del grupo. En caso de realizar la práctica en solitario, el formato será **ssoo-p1-NIA.zip**. **El archivo zip se entregará en el entregador correspondiente al código de la práctica**. El archivo debe contener:

- **Makefile**
- **crear.c**
- **combine.c**
- **autores.txt:** Fichero de texto en formato csv con un autor por línea. El formato es: NIA, Apellidos, Nombre

**NOTA**

Para comprimir dichos ficheros y ser procesados de forma correcta por el probador proporcionado, se recomienda utilizar el siguiente comando:

```
$ zip ssqo-p1-NIA1-NIA2-NIA3.zip Makefile crear.c combine.c autores.txt
```

La memoria se entregará en formato PDF en un fichero llamado:

**Ssoo\_p1\_NIA1\_NIA2\_NIA3.pdf**

Solo se corregirán y calificarán memorias en formato pdf. Tendrá que contener al menos los siguientes apartados:

- **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 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:
  1. Que un programa compile correctamente y sin advertencias (warnings) no es garantía de que funcione correctamente.
  2. 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.
- **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 archivo pdf se entregará en el entregador correspondiente a la memoria de la práctica (entregador TURNITIN).**

**NOTA:** Es posible entregar el código de la práctica tantas veces como se quiera dentro del plazo de entrega, siendo la última entrega realizada la versión definitiva.

**Importante:** Si se usan herramientas de AI y las soluciones de varios grupos coinciden, se considerará copia.

**LA MEMORIA DE LA PRACTICA ÚNICAMENTE SE PODRA ENTREGAR UNA UNICA VEZ A TRAVES DE**

**TURNITIN**

## 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. **Los programas que utilicen funciones de biblioteca (fopen, fread, fwrite, etc.) o similares, en vez de llamadas al sistema, obtendrán una calificación de 0. Tampoco se permite utilizar sentencias o funciones como goto o stat.**
3. 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.
4. Los programas deben compilar sin **warnings**.
5. 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 máquina virtual con Ubuntu Linux o en las Aulas Virtuales proporcionadas por el laboratorio de informática de la universidad. Si el código presentado no compila o no funciona sobre estas plataformas la implementación no se considerará correcta.
6. Un programa no comentado, obtendrá una **calificación muy baja**.

7. 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.
8. Se debe respetar en todo momento el formato de la entrada y salida que se indica en cada programa a implementar.
9. Se debe realizar un control de errores en cada uno de los programas, **más exhaustivo de lo solicitado explícitamente en cada apartado.**

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

## 4 Anexo (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 = función 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).

### 4.1 Llamadas al sistema relacionadas con archivos

*int open(const char \* path, int flag, ...)*

Abre o crea un fichero especificado por path. El fichero abierto puede utilizarse para lectura, escritura, o ambas, en función de lo especificado por flag. Devuelve un descriptor de fichero que se puede utilizar para la lectura o escritura en el archivo. Más ayuda en: man 2 open

*int close(int fildes)*

Cierra un archivo abierto anteriormente asociado al descriptor fildes. Si n = -1 → Error al cerrar el fichero. Más ayuda en: man 2 close

`ssize_t read(int fildes, void *buf, size_t nbytes)`

Intenta leer de un archivo (cuyo descriptor de fichero *fildes* se obtuvo de abrirlo) tantos bytes como indica *nbytes*, colocando la información leída a partir de la dirección de memoria *buf*. Devuelve el número de bytes leídos (que puede ser menor o igual a *nbytes*). Si retorno = 0 → Fin de fichero (EOF).

Si retorno = -1 → Error de lectura. Más ayuda en: man 2 read

`ssize_t write(int fildes, const void *buf, size_t nbytes)`

Intenta escribir en un archivo (cuyo descriptor de fichero *fildes* se obtuvo de abrirlo) tantos bytes como indica *nbytes*, tomándolos de la dirección de memoria indicada *buf*. Devuelve el número de bytes que realmente se han escrito (que puede ser menor o igual a *nbytes*). Si retorno = -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 archivo del siguiente write o read. Más ayuda en: man 2 write.

`off_t lseek(int fildes, off_t offset, int whence)`

Modifica el valor del apuntador del descriptor *fildes* en el archivo, a la posición explícita en desplazamiento a partir de la referencia impuesta en origen, de forma que las llamadas write o read pueden iniciarse en cualquier parte del archivo.

- Si retorno = -1 → Error de posicionamiento.

El parámetro *whence* 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 offset 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

## 4.2 Manual (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 [sección] open**

Una página de manual tiene varias partes. Estas están etiquetadas como NOMBRE, SINOPSIS, DESCRIPCION, OPCIONES, FICHEROS, .... 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. **Para salir de la página mostrada, basta con pulsar la tecla 'q'.** 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.

## 5 Bibliografía

- El lenguaje de programación C: diseño e implementación de programas éelix García, Jesús Carretero, Javier Fernández y Alejandro Calderón. Prentice-Hall, 2002.
- N. Matthew and R. Stones. Programación Linux. Anaya Multimedia. 2008. ISBN: 978-8441524422
- Leguaje C. <https://en.cppreference.com/w/c>
- POSIX. <https://pubs.opengroup.org/onlinepubs/9799919799/functions/contents.html>
- Sistemas Operativos: Una visión aplicada Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez. McGraw-Hill, 2001.
- Unix man pages (man function)