Universidad Carlos III de Madrid
Departamento de Informática
Área de Arquitectura y Tecnología de Computadores

Grado en Ingeniera Informática
SISTEMAS OPERATIVOS

Práctica 3.

Llamadas al sistema de ficheros

Índice

1 Enunciado de la Práctica

Esta práctica permite al alumno familiarizarse las llamadas al sistema POSIX.

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 las llamadas a funciones.

Este capítulo describe el enunciado de la práctica, el plazo y la forma de entrega.

Un aspecto muy importante para la correcta realización de las prácticas es respetar en todo momento el formato de la entrada y salida que se indica. Para ello se recomienda prestar mucha atención a los ejemplos que se ponen en cada uno de los enunciados. Asimismo, se recomienda respetar el formato de entrega de cada una de las prácticas.

Un segundo aspecto muy importante que hay que tener en cuenta en todas las prácticas, es el uso de 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.

Así, siguiendo estas mínimas reglas, 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.

2 Descripción de la Práctica

Se pretende programar un conjunto de herramientas que permitan el manejo de datos de alumnos recogidos en ficheros binarios. El fichero binario posee los siguientes campos por alumno (seguidos por el tamaño de ocupan):

- Nombre y apellidos [52 bytes]
- Nota [4 bytes]: valor entero comprendido entre 0 y 10.
- Convocatoria [4 byte]: valor comprendido entre 1 y 6.

Los distintos programas a codificar son los siguientes:

2.1 bin2text

Permite convertir un fichero de alumnos binario en otro de texto. En el fichero de texto, cada línea representa a un alumno, mientras que los datos de un alumno en se encontrarán delimitados por un tabulador ('\t').

Uso: `./bin2text <fichero binario> <fichero de texto>`

Nota: para convertir cadenas de caracteres en valores numéricos ver la función `atoi` (`man atoi`)

Nota: para acceder a los datos del fichero de texto se pueden usar las funciones de manejo de ficheros como `fprintf`, `fscanf`, etc. Para el fichero binario, solo se pueden usar llamadas POSIX: `write, read, open, etc.`

2.2 text2bin

Permite convertir un fichero de alumnos de texto en otro binario.

Uso: `./text2bin <fichero de texto> <fichero binario>`

Nota: para convertir cadenas de caracteres en valores numéricos ver la función `atoi (man atoi)`

Nota: para acceder a los datos del fichero de texto se pueden usar las funciones de manejo de ficheros como `fprintf`, `fscanf`, etc. Para el fichero binario, solo se pueden usar llamadas POSIX: `write, read, open, etc.`

2.3 statistics

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

- Porcentaje de alumnos con matrícula de honor (M): Nota = 10.
- Porcentaje de alumnos con sobresaliente (S): Nota = 9.
- Porcentaje de alumnos con notable (N): Nota = 7 o 8.
- Porcentaje de alumnos con aprobado (A): Nota = 5 o 6.
- Porcentaje de alumnos suspensos (s): Nota < 5.

Para la práctica, el porcentaje no tiene decimales. Ejemplo:

```
M:   1%
S:   9%
N:   20%
A:   50%
s:   20%
```

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

```
M:   < porcentaje de M >%
S:   < porcentaje de S >%
N:   < porcentaje de N >%
A:   < porcentaje de A >%
s:   < porcentaje de s >%
```

2.4 combine

Programa que combina datos de dos ficheros binarios, almacenando la combinación en un tercero pasado por parámetro.

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

2.5 split

Programa que divide en dos los datos almacenados en un fichero, generando dos ficheros de igual número de alumnos (o con una diferencia de un en caso de que el número de alumnos almacenados en el fichero origen sea impar que irá a parar al segundo fichero de salida).

Uso: `./split <fichero binario de entrada> <fichero binario de salida 1> <fichero binario de salida 2>`

2.6 *filter*

Este programa filtra a los alumnos según su nota, generando un fichero de salida como resultado con los alumnos que cumplen el filtro. El filtro se representa mediante una letra [M, S, N, A, s, a]:

- M: tiene como valor 10.
- S: tiene como valor 9.
- N: 8-7.
- A: 6-5.
- a: valores mayores o iguales a 5.
- s: valores menores de 5.

Uso: `./filter [letra del filtro] <fichero binario de entrada> <fichero binario de salida>`

2.7 *Código Fuente de Apoyo*

Para facilitar la realización de esta práctica se dispone del fichero llamadas.2009.tgz que contiene código fuente de apoyo. Al extraer su contenido desde un directorio de vuestra cuenta, se crea el directorio *ssoo/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.

bin2text.c

Fichero fuente de C que muestra cómo usar el programa bin2text.

text2bin.c

Fichero fuente de C que muestra cómo usar el programa text2bin.

statistics.c

Fichero fuente de C que muestra cómo usar el programa statistics.

combine.c

Fichero fuente de C que muestra cómo usar el programa combine.

split.c

Fichero fuente de C que muestra cómo usar el programa split.

filter.c

Fichero fuente de C que muestra cómo usar el programa filter.

3 Documentación a Entregar

Los ficheros a entregar de forma electrónica serán los siguientes:

- **memoria.pdf**

Memoria de la práctica (véase Normas de Presentación Generales).

- **practica3.2009.zip**

Que contiene los siguientes ficheros:

  - **bin2text.c**
  - **text2bin.c**
  - **statistics.c**
  - **combine.c**
  - **split.c**
  - **filter.c**

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

4 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 Mycrosystems Sun Mycrosystems, 1990.

5 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. Para obtener información sobre cualquiera de las llamadas, se puede utilizar el comando

```
man (man funcion_estandar).
```

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`).

5.1 Llamadas al sistema relacionadas con archivos

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

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

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

Ejemplo: derechos = 7 1 0 (octal) → binario→**r w x r w x r w x** →1 1 1 0 0 1 0 0 0

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

Abre un archivo existente. El modo determina si se abre para escritura, lectura o ambas. Modo: **0** - Lectura. **1** - Escritura. **2**- L/E.

Devuelve un descriptor de fichero que se puede utilizar para la lectura o escritura en el archivo. Si dos procesos distintos abren el mismo fichero, cada uno tendrá un descriptor de archivo distinto. Esto se debe a que cada proceso tiene una tabla de archivos abiertos (TAD).

```
n = read(fd, direccion_mem, nº_bytes)
```

Lee de un archivo (cuyo descriptor de fichero se obtuvo de abrirlo) tantos bytes como indica `nº_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 `nº_bytes`. Si n = 0 →Fin de fichero (EOF). Si n = -1→ Error de lectura.

```
n = write(fd, direccion_mem, nº_bytes)
```

Escribe en un archivo (cuyo descriptor de fichero `fd` se obtuvo al abrirlo) tantos bytes como indica `nº_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 `nº_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 archivo del siguiente `write` o `read`.

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

Modifica el valor del apuntador de desplazamiento en el archivo, 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 archivo.

Si x = -1 →Error de posicionamiento.

Origen:

- 0 - principio del fichero.
- 1 - posición actual.
- 2 - final del fichero.

Ejemplo:

```
a b c d e f g h i
lseek (4,4,0) → Se colocara en la "d".
```

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

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