UNIVERSITAT DE BARCELONA

Sistemas Operativos II

PRÁCTICA 3

30 de octubre 2020

Autores y NIUB:
María Isabel González Sánchez 20221913
Oriol Saguillo González 20150502

Grupo: 18, B00

Índice

1. Experiments amb la manipulació de fitxers

1.1. Crides a sistema

1. El tamaño del fichero después de ejecutarlo debería ser 4N, debido a que un int se almacena en 4 bytes y le hemos pedido N enteros. Por lo tanto, tenemos los siguientes casos para el tamaño del fichero:

- **Durante la ejecución del código:** tendrá una medida de 4N bytes.
- **Después de la ejecución:** será la misma que en el paso anterior, porque va directamente a Sistema, es decir, no pasa por el buffer de escritura.

Por ejemplo, primero hemos escrito N=7 ints (tamaño: 4N=4x7=28 bytes) y luego, N=3. Por lo tanto, pasaremos de 28 a 12 bytes porque se reinicia el fichero con la nueva ejecución. Esta sería un ejemplo de ejecución:

oslab: /Escritorio/codi$ ./write_int ejemplo_int.txt 3

Check the file! 3 integers have been written.

Press Enter to close the file

Closing the file

2. La llamada a sistema *write* escribe estos N ints que hemos pedido como parámetro en formato binario. Aquí surge un problema: el editor de texto plano debería volver a leerlos en binario, pero no puede porque no está especificada la encodificación a binario. Solo reinterpretará estos ints como caracteres inteligibles, por lo que se verá un resultado "extraño".

3. La diferencia entre un editor de texto plano y la aplicación Ghex es que Ghex lee nuestros N ints en formato binario y los muestra en formato octal, hexadecimal y ASCII, no como el editor que solo genera caracteres extraños. Por lo tanto, Ghex muestra en su segunda columna los N ints así: XX XX XX XX por int (4 bytes), donde las X son caracteres en hexadecimal.

4. En el código *read_int.c*, tenemos la llamada a sistema *read*, que es capaz de leer y traducir los N ints escritos en binario. Por ello, tras leer el contenido del fichero (en binario), lo traducirá a un formato legible en la consola. Un ejemplo sería escribir primero N=3 ints con *write_int.c* en un fichero y luego leerlos:

oslab: /Escritorio/codi$ ./read_int ejemplo_int.txt

0

1

2

5. Ahora estudiaremos los resultados de la ejecución del código *write_char_int.c*. Consiste en escribir el char "so2" N veces, siendo N el parámetro que nosotros

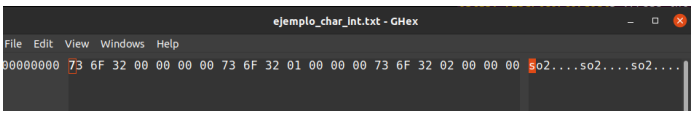
escogemos junto al fichero a escribir. Además, escribimos N ints de la misma forma que lo hace el código write_int.c. En cuanto al tamaño del fichero, tenemos 2 casos:

- **Durante la ejecución el código:** tendrá una medida de (4N+3N) bytes.
- **Durante y después de la ejecución:** será la misma que en el paso anterior, porque va directamente a Sistema, es decir, no pasa por el buffer de escritura.

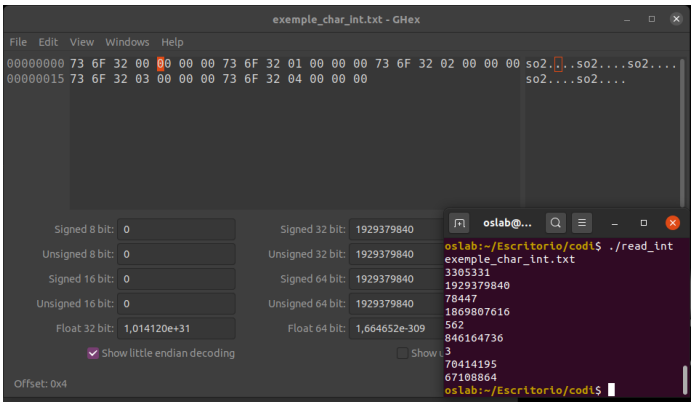
Para calcular dicha medida debemos saber que el tamaño del fichero será: (3N + 4N), es decir, la suma de los chars (se almacenan en 3 bytes) y los ints (en 4 bytes) escritos.

Pongamos un ejemplo. Queremos una N=3, por lo tanto nuestro tamaño será de 3N + 4N = 9 + 12 = 21 bytes.

6. Efectivamente, se escribe 3 veces el char "so2" junto con 3 ints intercalados en hexadecimal. Si N fuese 7, tendríamos 7 chars "so2", 7 ints y un fichero de 49 bytes. Así sucesivamente.



7. En este caso, la aplicación no peta pero los valores son incorrectos. Esto se debe a que en el código *read_int.c* tenemos la llamada a sistema *read*, que lee todo el contenido de nuestro fichero en binario como si fuesen solo ints, cuando también hay chars "so2". Entonces sigue un patrón: como un int está almacenado en 4B = 4 x 2^3 = 32 bits, saltará de 32 en 32 bits. Por ejemplo, esto se puede ser en la siguiente imagen:



Primero, cogerá el 73 en hexadecimal y lo pasará a un entero de 3305331. Luego se desplazará 4 bytes sucesivamente para coger los siguientes hasta exponer el número de enteros que haya: 00 por 1929379840, 6F por 78447, etc.

1.2. Funcions de la llibreria d'usuari

1. El tamaño del fichero debería ser 4N, ya que un int se almacena en 4 bytes y le hemos pedido N enteros (siendo N = {10,100,1000,2000} por ejemplo). Por lo tanto, tenemos los siguientes casos para el tamaño del fichero:

- **Durante la ejecución del código:** tendrá una medida de 0 bytes. Esto es debido a que, a diferencia de la llamada a sistema *write()*, *fwrite()* no va directamente al sistema, sino que antes pasa por un buffer interno.
- **Después de la ejecución:** será 4N bytes, porque el contenido del buffer ya ha pasado al sistema.

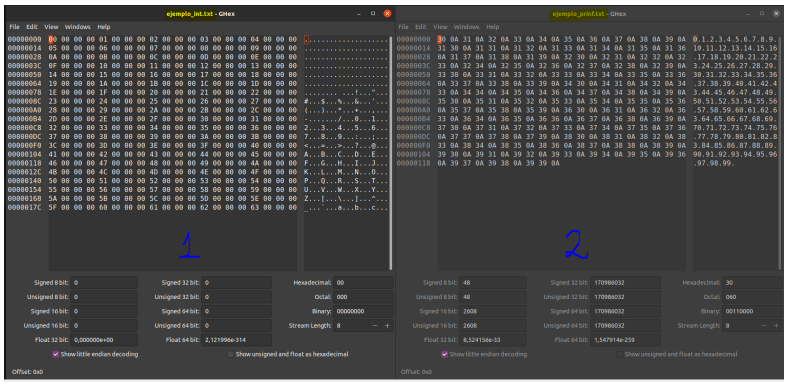
Por ejemplo, con N = {10,100,1000,2000} tendremos respectivamente unos tamaños de 40B, 400B, 4kB y 8kB.

2. El tamaño que tendrá el fichero se puede saber de forma teórica. Sea N el número de ints que queremos escribir, sabemos que cada entero se almacena en 4 bytes. Por lo tanto, el tamaño final de nuestro archivo será: 4 x N (bytes).

3. Se leerán los mismos números tanto con *read()* como con *fread()*. La diferencia es que *read()* llama directamente al sistema sin pasar por un buffer interno; mientras que la función *fread()*, antes de mostrar los datos por pantalla, los lee del buffer interno y si no están llama al sistema para rellenar dicho buffer con *read()*. Por lo tanto, la lectura con *read()* es ligeramente más rápida.

4. También leerán los mismos datos tanto *read()* como *fread()*. Debido a lo expuesto en el apartado 3, la diferencia reside en el tiempo que se tardarán en leer y mostrar por pantalla.

5.



Como podemos ver, con la función *fprinf()* (foto parte 2) se han escrito los 100 números en ASCII separados por puntos (0A hexadecimal)y empezando por el 30 Hexadecimal = 0 ASCII. Por ejemplo, el .10. ASCII está representado en hexadecimal como 0A 31 30 0A. Por otro lado, con *write()* (foto parte 1), los ints escritos en el fichero comienzan en el 01h (carácter de control en ASCII) y se van aumentado. Según avanzamos en el fichero, empiezan a aparecer caracteres ASCII inteligibles como "!" (21), "&" (26) o los números de una cifra.

6. La llamada sistema *write()* es de bajo nivel y no tiene las capacidades de formateo de datos que tiene la función *fprintf()*. El editor de texto no es capaz de expresar bien el contenido del fichero si está en binario, que es lo que le pasa con write_int.c. En cambio, *fprintf()* está diseñada para salidas formateadas, es decir, dicha salida tendrá un formato de texto "legible por humanos". En otras palabras, *fprintf()* es capaz de escribir en el fichero lo que nos saldría por consola si usásemos la llamada a sistema *read()*.

7. Sí, se pueden leer y mostrar correctamente en la consola debido a que, previamente con la función *fprintf()*, ya se ha cambiado la encodificación del fichero de binario a hexadecimal y la equivalencia a ASCII.

8. Ocurre como en el 7 del bloque 1: lee todo el contenido del fichero que ya está en ASCII e intenta traducirlo otra vez a un lenguaje legible. Además, sigue el patrón de avanzar la lectura de 4 en 4 bytes, de ahí que salgan los caracteres extraños: coge los valores del fichero como si estuviesen en binario y los traduce otra vez dando lugar a números como 170986032 para el 0 o 171117106 para el 2.