

Problema 2.1

ESCRIURE_ERROR

Escriure una funció en llenguatge C anomenada `escriure_error` que escrigui pel canal de error estàndar un missatge d'error. El prototipus d'aquesta funció serà:

```
void escriure_error(char *missatge, int tipus, int fi)
```

on el paràmetre `tipus` indica si es tracta d'un error propi de l'usuari o és un error associat a una crida a sistema. En el primer cas, treurà el missatge que l'usuari li passi com a primer paràmetre, mentre que en el segon cas imprimirà el missatge d'explicació que ofereix el sistema (utilitzeu la rutina `strerror`). El paràmetre `fi` indica si el programa ha d'acabar o no.

Aquesta serà la rutina que fareu servir quan us faci falta en la resta de problemes i en les pràctiques de l'assignatura. Exemples d'utilització son:

```
#define MIN_NUM 1
#define SISTEMA 1
#define PROPI 0
#define TRUE 1
#define FALSE 0
...
if ((fd = open("file.dat", O_RDONLY)) < 0)
    escriure_error(" ", SISTEMA, TRUE);
...
if (num < MIN_NUM)
    escriure_error("Valor de num massa petit", PROPI, FALSE);
```

Si feu **man** `strerror`:

C Library Functions

`strerror(3C)`

NAME

`strerror` - get error message string

SYNOPSIS

```
#include <string.h>
```

```
char *strerror(int errnum);
```

DESCRIPTION

The `strerror()` function maps the error number in `errnum` to an error message string, and returns a pointer to that string. It uses the same set of error messages as `perror(3C)`. The returned string should not be overwritten.

RETURN VALUES

The `strerror()` function returns `NULL` if `errnum` is out-of-range.

Modifiqueu el programa MICAT, que apareix tot seguit, de manera que no treballi llegint caràcter a caràcter sino utilitzant un buffer d’N caràcters.

```
main()
{ char c;

    while (read(0,&c,sizeof(char)) >0)
        write(1,&c,sizeof(char));
}
```

Quina avantatge creus que pot suposar treballar amb més d’un caràcter cada cop que es fa una crida a sistema?

Quin et sembla que seria el tamany aconsellable del buffer?

Problema 2.3**CATAIL**

Programeu un programa, que anomenarem **CATAIL**, modificant el programa MICAT original que teniu al problema 2.2. Aquesta nova versió sempre requereix com a paràmetre un fitxer (i només un) que serà el fitxer d’entrada; la sortida countinuarà fent-la pel canal de sortida estàndar. Des de la línia de comandes:

```
> catail mi_fitxer
```

El programa **CATAIL** no finalitza quan arribi al final del fitxer d’entrada: si el fitxer creix en qualsevol moment, es segueix copiant el seu contingut. L’execució del programa **CATAIL** s’haurà de fer explícitament des de la Shell mitjançant ctr-C.

Problema 2.4**MICP**

Feu un programa en C i crides al sistema de UNIX que permeti copiar fitxers plans (no directoris, ni fitxers especials), de manera que detecti els mateixos errors (lectura, escriptura, existència dels fitxers, etc...) i generi els mateixos missatges d’error que la comanda cp de UNIX.

```
> micp fitxer_origen fitxer_copiat
```

Heu d’afegir al vostre programa **MICP** una opció (**-o**) que indiqui que si el fitxer destí ja existeix no ha de sobre escriure’l.

Problema 2.5**REV**

Modifiqueu el comportament de **MICP** de manera que el fitxer copiat sigui l’invers. Aquest programa l’anomenarem **REV** i mantindrà la opció **-o** que heu afegit.

```
> rev fitxer_origen fitxer_invertit
```

Escribiu un programa (que anomenarem **MITEE**) que copiï el que rep per la seva entrada estàndar sobre la seva sortida estàndar i, a més a més, sobre un fitxer que ens passaran com a paràmetre. En el cas de que el fitxer que passem com a paràmetre ja existeixi, **MITEE** ha de sobrescriure'l. Exemples:

```
> mitee
```

Donarà error ja que falta el paràmetre (el nom del fitxer).

```
> mitee pppp
```

Copiarà el que introduïm pel teclat sobre el fitxer pppp i, a més a més, ens ho mostrarà per pantalla.

```
> mitee cccc <aaaa >bbbb
```

Generarà dos fitxers, el bbbb i el cccc, tots dos amb el mateix contingut que aaaa. Explica com arriba a aquest resultat (d'on agafa l'informació, i per què i com crea els diferents fitxers).

Mira lo que fa l'utilitat tee (fes un man tee a moonrey) i afegeix a **MITEE** la mateixa opció **-a**.

Problema 2.7

CRIBADO

Escribe un programa en C con llamadas al sistema de UNIX que haga lo siguiente:

- Lee un carácter de la entrada estándar (`stdin`): podrá ser el carácter `I` (indica que lo que se leerá a continuación es un número de tipo `int`) o el carácter `F` (indica que lo que se leerá a continuación es un número de tipo `float`).
- Lee dos valores de la `stdin`: `int` o `float`, según el carácter leído y estarán en formato interno de nuestra máquina (no ASCII y, por tanto, no legible desde el terminal).
- Los suma y escribe el resultado, en formato interno también, en un fichero de salida de enteros o uno de coma flotante, según sea el caso. El nombre de estos dos ficheros se pasan por parámetro.

Esta operación la realizará mientras haya datos para tratar. La línea de comando será:

```
> cribado <tripletas sum_enteros sum_reales
```

Añade al programa anterior una opción (**-i**) para que al final muestre por la salida estándar, en formato ASCII, los valores en posiciones pares (0,2,4,6, etc.) del fichero de enteros.

Escribid un programa que codifique un fichero ASCII de entrada. Para hacerlo se dispone de un segundo fichero de control formado por parejas de valores enteros (pos,lon), en formato binario.

El algoritmo a seguir es:

```
Mientras hay datos en el fichero de control hacer
    leer pos y lon.
    posicionar el fichero de entrada en pos
    leer lon bytes del fichero de entrada
    escribir lon bytes en el fichero de salida
fmientras
```

Tened en cuenta las siguientes consideraciones:

- El fichero de salida será la salida estándar.
- El fichero de entrada será el primer parámetro del programa y el fichero de control, el segundo.
- El fichero de control está bien calculado y no se deja trozos del fichero original ni contiene posiciones fuera del fichero de entrada.
- “lon” puede ser mayor que la longitud del *buffer* (vector de caracteres) que se utilice para leer la información.

La línea de comando de este programa podría ser:

```
> cifrador >f_resultado f_fuente f_control
```

Problema 2.9

INTERCAMBIO DE PAREJAS

Implementar el comando `exchange` que intercambia dos columnas de caracteres de un fichero. La forma de invocar este comando es la siguiente:


```
> exchange c1 c2 file
```

Donde `file` será el fichero de entrada y `c1` y `c2` son las dos columnas a intercambiar. El fichero modificado se ha de extraer por la salida estándar. Un ejemplo:

```
> cat fichero.txt
ABCDEFGHI
123456789
```

Una posible ejecución sería:

```
> exchange 3 6 fichero.txt
ABCGEFDHI
123756489
```



Tened en cuenta las siguientes consideraciones:

- El final de una línea viene indicado por el carácter ‘\n’.
- Comprobad que las líneas del fichero tienen tantos caracteres como indiquen `c1` y `c2`.