Laboratorio de Arquitectura de Redes

Entrada y salida por archivos en lenguaje C

Entrada y salida por archivos lenguaje C

- Archivos y secuencias en lenguaje C
- Apertura y cierre de un archivo
  - Fin de fichero
- Entrada y salida de texto
  - Entrada y salida de caracteres
  - Entrada y salida de cadenas
- Lectura y escritura de datos binarios
- Entrada y salida con formato
- Entrada y salida mediante acceso directo
- Otras operaciones sobre archivos
  - Función `ftell()`
  - Función `rewind()`
  - Función `remove()`
  - Función `fflush()`
  - Función `tmpfile()`

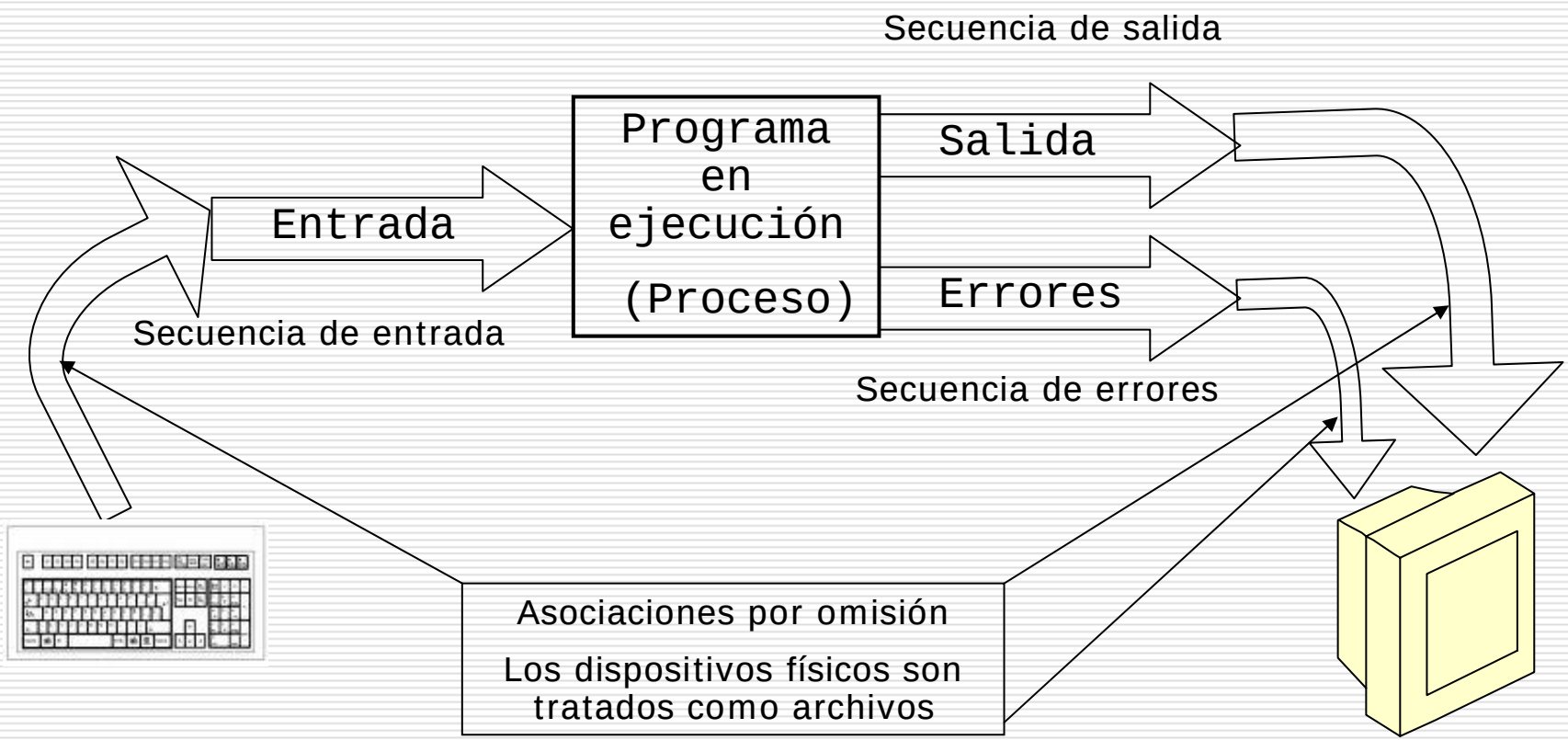
Archivos y secuencias en lenguaje C (I)

- El lenguaje C proporciona mecanismos para la entrada y salida por archivos
  - Que no dependen del dispositivo físico sobre el que actúen
  - Son funciones genéricas, potentes y flexibles
  - Distingue dos tipos de elementos
    - *Archivos*
    - *Secuencias* o *streams*
- **Archivo**: Dispositivo físico y real sobre el que se realizan las operaciones de entrada o salida
- **Secuencia** (**stream**): Ente abstracto sobre el que se realiza una operación de entrada o salida genérica
  - Las secuencias, en muchas ocasiones, se asocian a dispositivos físicos o a archivos:
    - La *secuencia de salida estándar* se asocia a la pantalla
    - La *secuencia de entrada estándar* se asocia al teclado

Archivos y secuencias en lenguaje C (II)

- El concepto de *archivo* puede aplicarse, además de a ficheros en disco, a impresoras, terminales, controladores, etc.
- El concepto de *secuencia* puede entenderse como un buffer de datos o un dispositivo lógico que proporciona un camino para los datos en las operaciones de entrada o salida
- Cuando se abre un archivo para operaciones de entrada y/o salida, se le asocia una secuencia, un camino que permitirá la transferencia de información entre ese archivo y el programa.
- El lenguaje C contempla los archivos como secuencias continuas de bytes que pueden ser accedidos de forma individual:
  - Todas las funciones de E/S sirven para todas las secuencias, sólo hay que redirigirlas al archivo deseado

Archivos y secuencias en lenguaje C (III)



Archivos y secuencias en lenguaje C (IV)

- El sistema operativo (DOS/Windows) admite que un archivo pueda ser abierto de dos modos:
  - En *modo texto:* Los bytes de la secuencia son códigos ASCII
  - En *modo binario*: Los bytes de la secuencia son códigos binarios de 8 bits
- En los sistemas Unix y Linux no existe tal distinción
  - No hay ninguna indicación añadida al modo de apertura de un archivo

Archivos y secuencias en lenguaje C (V)

- En *modo texto* el final de la línea se representa
  - En lenguaje C: como ‘\n’ (En ASCII CR: Retorno de carro, código 10)
  - En el sistema operativo: como la sucesión de los códigos ASCII: CR+LR (Retorno de carro+Salto de línea, códigos 10 y 13)
  - Las funciones de lectura en lenguaje C al encontrar la secuencia CR+LF, la convierten a ‘\n’
  - Las funciones de lectura al encontrar el código correspondiente a Ctrl-Z lo interpretan como EOF (*End of file*, fin de fichero).

Archivos y secuencias en lenguaje C (VI)

- Ejemplo: Si en un archivo de texto del DOS (`*.TXT`) escribimos

  ```
  En un lugar de la mancha,
  de cuyo nombre no quiero acordarme.
  ```

- Al leer el archivo en modo binario desde un programa en lenguaje C se obtendrá la siguiente secencia

  ```
  En un lugar de la mancha, \r\nde cuyo nombre no quiero acordarme.EOF
  ```

- Si se abre en modo texto, lo que se recogerá será lo siguiente:

  ```
  En un lugar de la mancha, \nde cuyo nombre no quiero acordarme.EOF
  ```

Archivos y secuencias en lenguaje C (VII)

- Al ejecutar un programa en modo consola, automáticamente el sistema operativo abre tres secuencias estándar:
  - `stdin`
    - Entrada estándar. Se asocia al teclado
  - `stdout`
    - Salida estándar. Se asocia a la pantalla
  - `stderr`
    - Salida estándar de errores. Se asocia a la pantalla
- Las secuencias estándar tienen asociado un buffer de memoria física cada una
- Las E/S por consola operan igual que las E/S por archivos, pero redirigen la operación real sobre las secuencias estándar `stdin`, `stdout` y `stderr`

Apertura y cierre de un archivo (I)

- En lenguaje C, para acceder a los archivos se precisa un **puntero a un archivo** (puntero de tipo `FILE`)
  - Declaración:

    ```
    FILE *puntero;
    ```

  - `FILE` es una constante definida en `STDIO.H`
  - El puntero apunta a un buffer que contiene toda la información necesaria para operar con el archivo
  - Se utiliza para hacer referencia al archivo en todas las operaciones sobre este
  - Debe declararse antes de utilizarse
  - El puntero se inicializa al abrir sin error un archivo al que se le hace apuntar

Apertura y cierre de un archivo (II)

- Antes de cualquier operación sobre un archivo es preciso **abrir el archivo**, utilizando la función `fopen()`
  - Declaración:

    ```
    FILE * fopen(char *nombrearchivo, char *modo);
    ```

    - Devuelve
      - Un puntero de tipo `FILE` que apunta al archivo abierto
      - Un puntero nulo (`NULL`) en caso de error
    - Recibe dos cadenas de caracteres:
      - La primera con el nombre del archivo (con la ruta de acceso)
        - En la ruta, las barras de separación de directorios deben escribirse dobles (`\\`)
      - La segunda con el modo de apertura que se selecciona

Apertura y cierre de un archivo (III)

MODOS DE APERTURA DE LOS ARCHIVOS EN LENGUAJE C			
Modo de apertura	**Cadena de modo de apertura**		**Observaciones**
	Modo texto	**Modo binario**	
Abrir para leer	"r"	"rb"	Si no existe, se produce error
Crear para escribir	"w"	"wb"	Si existe, se pierde el contenido
Abrir o crear para añadir	"a"	"ab"	Si no existe, se crea
Abrir para leer y/o escribir	"r+"	"rb+"	Debe existir
Crear para leer y/o escribir	"w+"	"wb+"	Si existe, se pierde el contenido
Abrir o crear para añadir y/o leer	"a+"	"ab+"	Si no existe, se crea

Apertura y cierre de un archivo (IV)

- Mientras el archivo esté abierto, el puntero correspondiente apunta a una estructura que contiene toda la información necesaria sobre el archivo: Nombre, tamaño, atributos, posición del apuntador para lecturas y/o escritura, etc.
- Al finalizar el programa, si termina normalmente, los archivos que estuviesen abiertos son cerrados por el sistema operativo.
- Un archivo no cerrado correctamente queda inutilizado y la información que pudiera contener se pierde y queda inaccesible
- En previsión de finalizaciones anómalas, tras las operaciones de lectura y/o escritura, es preciso cerrar los archivos, utilizando la función `fclose()`:
  - Declaración

    ```
    int fclose(FILE *puntero);
    ```

  - Devuelve
    - un entero de valor cero si el cierre ha sido correcto
    - EOF en caso de error
  - Recibe como argumento el puntero al fichero que se quiere cerrar

Apertura y cierre de un archivo (V)

- Ejemplo de apertura y cierre de un archivo:

```
FILE *pf;   /* Puntero a archivo */
if ((pf=fopen("D:\\MISDATOS\\PRUEBA.X","wb+")) == NULL)
{
   puts("\nNo es posible crear el archivo");
   exit(0);
}
else printf("\nEl archivo se ha abierto");
   /* Tras realizar todas las operaciones necesarias,
      es preciso cerrar el archivo antes de finalizar
      el programa */
fclose(pf);
   /* El archivo ha quedado cerrado */
```

Apertura y cierre de un archivo (VI)

- El **carácter de fin de fichero** está representado por la constante simbólica `EOF`, definida en `STDIO.H`
  - Es el último byte del archivo
  - Cuando se leen bytes del archivo (con `fgetc()`) puede leerse el `EOF` y no distinguirse como último carácter, especialmente cuando se trata de secuencias binarias
  - La función `feof()`, declarada en `STDIO.H` devuelve un valor distinto de cero (verdadero/true) cuando se lee el byte de fin de fichero (`EOF`)

```
while (!feof(puntfile))
{
   /* Operaciones sobre el archivo abierto */
}
```

Entrada y salida de texto (I)

- Las funciones de lectura y escritura de caracteres en archivos están definidas en `STDIO.H`

```
int fgetc( FILE *pf);
```

  - Lee un carácter en el archivo cuyo puntero recibe
  - Devuelve el carácter leído en un entero o `EOF` en caso de error

```
int fputc( int car, FILE *pf);
```

  - Escribe un carácter en el archivo cuyo puntero recibe
  - Devuelve `EOF` en caso de error
  - Recibe como argumentos:
    - `car`: El carácter a escribir
    - `pf`: El puntero al archivo

Entrada y salida de texto (II)

- Ejemplos de lectura y escritura de un carácter en un archivo:

```
FILE *pf1, *pf2;
char letra;
pf1 = fopen(“LEER.TXT”, “r”);
letra = fgetc(pf1);   /* Lee un carácter */
pf2 = fopen (“ESCRIBIR.TXT”, “w”);
fputc(letra, pf2);    /* Escribe un carácter */
fclose(pf1);
fclose(pf2);
```

Entrada y salida de texto (III)

- Las funciones de lectura y escritura de cadenas de texto en archivos están definidas en `STDIO.H`

```
char * fgets(char *cad, int numcar, FILE *pf);
```

  - Devuelve un puntero a la cadena leída o un puntero nulo en caso de error
  - Recibe como argumentos
    - `cad`: Un puntero a la zona de memoria donde se almacenará la cadena
    - `numcar-1`: Es el número de caracteres a leer. Será añadido el carácter nulo
    - `pf`: El puntero al archivo
  - Si encuentra un carácter de fin de línea (`\n`), éste será el último carácter leído

```
int fputs(char *puntcad, FILE *pf);
```

  - Devuelve en un entero el último carácter escrito o `EOF` en caso de error
  - Recibe como argumentos
    - `puntcad`: Un puntero a cadena que se quiere escribir
    - `pf`: El puntero al archivo

Entrada y salida de texto (IV)

- Ejemplos de lectura y escritura de una cadena de caracteres en un archivo:

```
FILE *pf1, *pf2;
char lect[50];
char escr[]="Mensaje a guardar en el archivo";
int num=50-1;
pf1 = fopen("LEER.TXT", "r");
fgets(lect, num, pf1);
   /* Lee una cadena de 49 caracteres de LEER.TXT */
pf2 = fopen ("ESCRIBIR.TXT", "w");
fputs(escr, pf2);
   /* Escribe la cadena "Mensaje a guardar en el archivo"
   en ESCRIBIR.TXT */
fclose(pf1);
fclose(pf2);
```

Lectura y escritura de datos binarios (I)

- Para la lectura de un conjunto de datos

```
unsigned fread(void *buf, unsigned numbytes,
               unsigned numdat, FILE *pf);
```

- Para la escritura de un conjunto de datos

```
unsigned fwrite(void *buf, unsigned numbytes,
                unsigned nundat, FILE *pf);
```

  - Devuelven el número de datos leídos o escritos
  - Reciben
    - `buf`: Un puntero a los datos que son leídos o escritos
    - `numbytes`: representa el número de bytes de que consta cada uno de los datos a a leer o escribir (se obtiene con sizeof)
    - `numdat`: representa el número total de datos, items o elementos a leer o escribir
    - `pf`: es el puntero al fichero sobre el que van a leer o escribir

Lectura y escritura de datos binarios (II)

- Mediante el manejo de datos binarios el contenido de los archivos puede ser similar al contenido de las variables en memoria
  - Es importante cuando se trabaja con estructuras y uniones
- Las funciones `fread` y `fwrite` pertenecen al ANSI C y están definidas en `STDIO.H`
- Ejemplo de utilización:

```
FILE *pf;
float valor1=3.5, valor2;
pf=fopen (“arhivo.dat”, “ab+”);
fwrite(&valor1, sizeof(valor), 1, pf); /*Escribe */
fread(&valor2, sizeof(float), 1, pf); /* Lee */
```

Entrada y salida por archivos con formato (I)

- Las funciones `fprintf()` y `fcanf()` son similares a printf() y scanf() respectivamente pero operan sobre archivos. También están declaradas en STDIO.H

```
int fprintf(FILE *pf, const char*cadcontrol, listargumentos);
int fscanf(FILE *pf, const char*cadcontrol, listargumentos);
```

  - Argumentos que reciben
    - `pf`: Representa un puntero al fichero sobre el que opera la función
    - `cadcontrol`: Es la cadena en la que s e incluyen los especificadores de formato y sus modificadores
    - `listargumentos`: Representa la lista de argumentos que se corresponden con los especificadores de formato de la cadena de control (lista de variables cuyos contenidos quieren escribirse o leerse sobre el archivo
  - Devuelven
    - `fprintf()` devuelve en un entero el número de bytes escritos
    - `fscanf()` devuelve en un entero el número de campos correctamente escaneados y almacenados o EOF en caso de error.
  - La llamada `fprintf(stdout, "Número: %d", num);` es equivalente a la llamada `printf("Número: %d", num);`

Entrada y salida por archivos con formato (II)

- Cuando se escribe en un archivo mediante la función `fprintf()` el contenido del archivo es similar al mostrado en pantalla con la función `fprintf()`
- Ejemplo: Escritura y lectura con formato

```
FILE *pf;
int i = 100;
char c = 'C';
float f = 1.234;
pf = fopen("PRUEBA.DAT", "w+");
fprintf(pf, "%d %c %f", i, c, f);
   /* Escribe en el archivo */
fscanf(pf, "%d %c %f", &i, &c, &f);
   /* Lee los mismos datos en el archivo */
fclose(pf);
```

Entrada y salida mediante acceso directo (I)

- El acceso a los archivos puede hacerse
  - En **modo secuencial**: los datos son leídos o escritos seguidos, sin saltos
  - En **modo aleatorio**: es posible acceder a cualquier posición para realizar una operación de lectura o de escritura
- Los punteros de tipo `FILE` apuntan a una estructura creada por el sistema operativo que controla las operaciones sobre ese archivo.
  - La estructura incluye un puntero de lecto-escritura que determina la posición actual en la que se va a escribir o a leer en cada momento
  - Al abrir un archivo, el puntero de lecto-escritura apunta al comienzo del archivo (salvo se si abre para añadir)
- La función `fseek()`, definida en `STDIO.H`, permite el movimiento aleatorio por el archivo abierto estableciendo una nueva posición para el apuntador de lecto-escritura

Entrada y salida mediante acceso directo (II)

`int fseek(FILE *pf, long nbytes, int origen);`

- Devuelve un valor verdadero si el movimiento se realizó con éxito o cero en caso contrario
- Recibe
  - `pf`: Puntero al archivo sobre el que se opera
  - `nbytes`: Número de bytes que queremos desplazar el apuntador de lecto-escritura del archivo con relación al punto de partida
  - `origen`: Representa el punto que se toma como origen o referencia. Se utilizan constantes simbólicas definidas en `STDIO.H`:
    - `SEEK_SET` corresponde al principio del archivo
    - `SEEK_CUR` corresponde a la posición actual
    - `SEEK_END` corresponde al final del archivo

Entrada y salida mediante acceso directo (III)

- Ejemplo:

```
#define N 5              /* Será la fila 5 */
int matriz[20][4], *punt;  /* 20 filas y 4 columnas*/
FILE *pf;
punt = matriz;
.../* Apertura del archivo sin errores */
fwrite(punt, sizeof(int) , 20*4, pf);
   /* Escribe la matriz en un archivo */
fseek(pf, sizeof(int)*4*N, SEEK_SET);
   /* Apunta a los datos de la fila N, al ser
      4*sizeof(int) el tamaño de una fila */
fread(&matriz[N][0], sizeof(int), 4, pf);
   /* Lee los datos de la fila N */
```

Otras operaciones sobre archivos (I)

- Función `ftell()`

  `long ftell(FILE *pf);`

  - Devuelve un entero largo con la posición del apuntador de lecto-esritura del archivo con respecto al principio del archivo.
  - Recibe el puntero a un archivo abierto
  - Está definida en `STDIO.H`

Otras operaciones sobre archivos (II)

- Función `rewind()`

  ```
  void rewind(FILE *pf);
  ```

  - Inicializa el indicador de posición o apuntador de lecto-escritura haciendo que apunte al principio del archivo
  - No devuelve nada
  - Recibe el puntero a un archivo abierto
  - Está definida en `STDIO.H`

Otras operaciones sobre archivos (III)

- Función `remove()`

  ```
  int remove(char *nombrearchivo);
  ```

  - Borra el archivo cuyo nombre se especifique en la cadena que recibe como argumento
  - Devuelve cero si la operación tuvo éxito y `-1` si se produce algún error
    - En caso de error, la variable global `errno`, definida en `ERRNO.H` indicará el tipo de error
  - Está definida en `STDIO.H`

Otras operaciones sobre archivos (IV)

- Función `fflush()`

  ```
  int fflush(FILE  *pf);
  ```

  - Vacía el contenido de una secuencia de entrada o salida, borrando toda la información almacenada en el buffer de memoria asociado a esa secuencia
  - Devuelve cero si la operación tuvo éxito y `NULL` si se produce algún error
  - Está definida en `STDIO.H`
  - Es muy utilizada para borrar el buffer del teclado y cuando se trabaja con impresoras

Otras operaciones sobre archivos (V)

- Función `tmpfile()`

```
FILE * tmpfile(void));
```

  - Crea un fichero temporal que es borrado automáticamente cuando el fichero es cerrado o al terminar el programa
  - El fichero temporal se crea en modo “`w+b`”
  - Devuelve un puntero al fichero temporal creado o un puntero nulo si no se puede crear
  - Está definida en `STDIO.H`