PRÁCTICA 9

SERVICIOS DE VOLUMEN, DIRECTORIO Y FICHEROS EN WINDOWS

Objetivos

El objetivo de esta práctica es escribir un programa que realice una copia de seguridad de los ficheros del directorio raiz de una unidad de disco en una unidad distinta. Para ello necesitaremos utilizar diversos servicios de Windows relacionados con el sistema de ficheros:

- Servicios de volumen: `GetLogicalDrives`.
- Servicios de directorio: `GetCurrentDirectory`, `SetCurrentDirectory`, `CreateDirectory`.
- Servicios de ficheros: `CopyFile`, `FindFirstFile`, `FindNextFile`, `FindClose`.

Cuestiones previas

Para utilizar los servicios de Windows es necesario incluir el fichero de cabecera `windows.h`.

Para realizar adecuadamente esta práctica es necesario haber repasado previamente algunos conceptos de C: reserva dinámica de memoria (función `malloc`), y utilización de operadores que trabajan con bits, en particular, AND binario (&) y máscaras.

1. Servicios de volumen

GetLogicalDrives

```
DWORD GetLogicalDrives(void);
```

Este servicio devuelve un valor de 32 bits en el que cada bit representa una unidad lógica. Por ejemplo, si el sistema tiene unidad A entonces el bit 0 estará activado (su valor será 1), y si el sistema tiene unidad C, estará activado el bit 2.

Ejercicio 1 Escribir un programa en C que muestre por pantalla las unidades lógicas asignadas. Además deberá mostrarse un mensaje indicando cuales son la primera unidad asignada y la última.
Ejemplo de salida por pantalla:

```
c:\ d:\ i:\ k:\
Primera unidad c:\
Ultima unidad k:\
```

Para resolver este ejercicio necesitarás utilizar el operador & (*AND binario*) y máscaras. La función `sprintf` también puede serte de utilidad.

2. Servicios de directorio

GetCurrentDirectory

```
DWORD GetCurrentDirectory( DWORD nBufferLength,
                           LPTSTR lpBuffer);
```

Este servicio rellena el buffer apuntado por `lpBuffer` con el directorio de trabajo actual del proceso. El servicio devuelve:

- 0 en caso de error.
- El número de caracteres copiados en el buffer, siempre y cuando este número sea menor o igual que `nBufferLength`.
- En caso de que el tamaño del buffer sea insuficiente, devuelve el tamaño mínimo que debería tener el buffer para contener la información pedida acerca del directorio. Este tamaño mínimo incluye el carácter de terminación de cadena \0

El servicio `GetCurrentDirectory` debe utilizarse de la siguiente manera:

```
(1)  nBufferLength=GetCurrentDirectory(0,NULL);
(2)  lpBuffer=(LPTSTR)malloc(nBufferLength);
(3)  tam=GetCurrentDirectory(nBufferLength,lpBuffer);
```

1. Inicialmente obtenemos el tamaño que es necesario reservar (longitud de la cadena más el carácter de terminación de cadena)
2. Reservamos el espacio y obtenemos el puntero a la zona reservada
3. Se copia la cadena en el espacio reservado y se obtiene (variable `tam`) el número de bytes copiados.

SetCurrentDirectory

```
BOOL SetCurrentDirectory(LPTSTR lpszCurdir);
```

Permite cambiar el directorio actual (directorio de trabajo) del proceso que lo ejecuta. El cambio de directorio actual sólo altera el directorio actual del proceso que realiza la llamada, y no el de los otros procesos en ejecución. Devuelve TRUE cuando la operación tiene éxito y FALSE cuando falla.

CreateDirectory

```
BOOL CreateDirectory( LPTSTR lpPathName,
                      LPSECURITY_ATTRIBUTES lpSecurityAttributes );
```

Este servicio permite crear un directorio. Devuelve TRUE cuando la operación tiene éxito y FALSE cuando falla.

En la creación, se pueden asignar privilegios al directorio utilizando la estructura `SECURITY_ATTRIBUTES`. Ésto solo es posible cuando se crea un directorio en una partición NTFS. En el resto de casos el segundo parámetro puede dejarse a 0.

Ejercicio 2 Sobre la base del ejercicio 1, mostrar por pantalla cuál es el directorio del cual se va realizar el backup (raiz de la última unidad asignada) y directorio de destino del backup, que tendrá un nombre de la forma `copia_X` (donde X es el nombre de la última unidad asignada) y deberá crearse en el raiz de la primera unidad. Finalmente habrá que cambiar el directorio de trabajo al nuevo directorio (con SetCurrentDirectory) y mostrarlo por pantalla (utilizando GetCurrentDirectory).
Ejemplo de salida por pantalla:

```
Directorio de origen k:\
Directorio de destino c:\copia_k\
Creado el directorio c:\copia_k\
Directorio de trabajo actual c:\copia_k\
```

3. Servicios de ficheros

FindFirstFile

```
HANDLE FindFirstFile( LPCTSTR  lpFileName,
                      LPWIN32_FIND_DATA lpFindFileData );
```

Permite buscar archivos y subdirectorios en un directorio determinado. El primer parámetro es una cadena que contiene un nombre de patrón de fichero. El segundo es un puntero a una estructura `WIN32_FIND_DATA` (que se muestra más adelante).

Si `FindFirstFile` tiene éxito en la localización de un archivo en el directorio que coincida con el patrón de fichero, rellena los campos de la estructura `WIN32_FIND_DATA` y devuelve un descriptor (tipo `HANDLE`). Si `FindFirstFile` no puede encontrar un fichero de esas características, devuelve `INVALID_HANDLE_VALUE` y no se cambia la estructura.

FindNextFile

```
BOOL FindNextFile( HANDLE  hFindFile,
                   LPWIN32_FIND_DATA  lpFindFileData );
```

Si `FindFirstFile` tiene éxito, se puede utilizar la función `FindNextFile` para que busque el siguiente fichero que coincida con el patrón indicado.

El parámetro `hFindFile` es el descriptor que devolvió la llamada anterior a `FindFirstFile`, y el parámetro `lpFindFileData` es, de nuevo, un puntero a una estructura `WIN32_FIND_DATA`.

Si `FindNextFile` tiene éxito devuelve TRUE y rellena la estructura `WIN32_FIND_DATA`.

FindClose

```
BOOL FindClose( HANDLE  hFindFile );
```

Cuando hayamos terminado de encontrar ficheros, debemos cerrar el descriptor devuelto por `FindFirstFile` llamando a `FindClose`.

Estructura de búsqueda

La estructura de un programa de utilice estos tres servicios para buscar los ficheros que cumplen un determinado patrón (en este ejemplo cualquier fichero ya que el patrón es `*.*`) es la siguiente:

```
HANDLE hFind
WIN32_FIND_DATA datos;

hFind = FindFirstFile( "*.*", &datos);

if (hFind != INVALID_HANDLE_VALUE)
{
   /* procesar fichero */
   ...
   while (FindNextFile(hFind,&datos))
   {
        /* procesar fichero */
   }
   FindClose(hFind);
}
```

Estructura `WIN32_FIND_DATA`

```
typedef struct _WIN32_FIND_DATA {
```

```
    DWORD dwFileAttributes;      // atributos
    FILETIME ftCreationTime;     // hora de creación
    FILETIME ftLastAccessTime;   // hora de último acceso
    FILETIME ftLastWriteTime;    // hora de última modificación
    DWORD    nFileSizeHigh;      // tamaño (bytes mas significativos)
    DWORD    nFileSizeLow;       // tamaño (bytes menos significativos)
    DWORD    dwReserved0;
    DWORD    dwReserved1;
    TCHAR    cFileName[ MAX_PATH ];     // nombre
    TCHAR    cAlternateFileName[ 14 ];  // nombre corto (estilo 8.3)
} WIN32_FIND_DATA;
```

Cómo utilizar los campos de la estructura:

- Atributos. El campo `dwFileAttributes` codifica en cada bit un atributo diferente. Por ejemplo, si queremos saber si el fichero es un directorio tenemos que comparar (AND binario) el campo de atributo con la máscara `FILE_ATTRIBUTE_DIRECTORY`.
- Tamaño. El tamaño se almacena en dos campos. El tamaño total se obtiene mediante:

  `nFileSizeHigh * MAXDWORD + nFileSizeLow`

- Las fechas se almacenan en formato UTC (coordinadas universales). Para poder tratarlas facilmente es necesario:
  - Transformarlas a hora local
  - Cambiar el formato de la información (`FILETIME` $\Longrightarrow$ `SYSTEMTIME`)

  Para ello se utilizan los servicios:
  - `FileTimeToLocalFileTime(FILETIME *, FILETIME *)`
  - `FileTimeToSystemTime(FILETIME *, SYSTEMTIME *)`

  La estructura `SYSTEMTIME` contiene los campos: `wDay`, `wMonth`, `wYear`, `wHour`, `wMinute`.

Ejercicio 3 Sobre la base del ejercicio 2, mostrar un listado de todos los ficheros (que no sean directorios) del directorio origen donde en cada línea aparezca la siguiente información:

`Fecha_ultima_modificacion Hora_ultima_modificacion Nombre_del_fichero`

CopyFile

```
BOOL CopyFile( LPCTSTR  lpExistingFileName,
               LPCTSTR lpNewFileName,
               BOOL  bFailIfExists );
```

Copia el archivo identificado por el parámetro `lpExistingFileName` en un archivo nuevo llamado `lpNewFileName`. El parámetro `bFailIfExists` especifica si se desea que la operación falle si ya existe un archivo con ese nombre (valor TRUE) o si se destruye el archivo existente y se crea el nuevo (valor FALSE).

Devuelve TRUE si la operación tuvo éxito. Falla si algún proceso tiene abierto el fichero existente con acceso de escritura.

Ejercicio 4 Sobre la base del ejercicio 3, copiar todos los ficheros (que no sean directorios) del directorio origen al directorio destino. Para cada fichero copiado debe mostrarse una línea que informe del nombre del fichero copiado. Finalmente debe mostrarse un listado del directorio de destino.