Una clase Archivo en C++

Elaborado por: Ukranio Coronilla

Debido a que el manejo de archivos es de mucha utilidad en el desarrollo de sistemas distribuidos, elaboraremos la clase Archivo para utilizarla en nuestras aplicaciones. Para ello revise el funcionamiento del código descrito en la práctica 8 “Manejo de archivos” del manual “Programación de Sistemas Linux”, y mediante la utilización únicamente de las llamadas al sistema `open`, `read`, y `write`; programe la implementación de la clase Archivo la cual va a modelar un archivo, y cuya interfaz se muestra a continuación.

```
class Archivo
{
private:
   string nombreArchivo;
   int fd;
   char *contenido;
   size_t num_bytes;

public:
   Archivo(string filename);
   Archivo(string filename, int banderas, mode_t modo);
   size_t lee(size_t nbytes);
   size_t escribe(void &buffer, size_t nbytes);
   size_t obtieneNum_bytes();
   const char *get_contenido();
   ~Archivo();
};
```

El objeto archivo almacena en `nombreArchivo` el nombre del archivo, en `fd` el descriptor de archivo correspondiente y en *`contenido` el contenido parcial o total del archivo. La variable privada `num_bytes` almacena el número de bytes que se encuentran almacenados dentro del objeto hasta ese momento.

Observe que un objeto Archivo se encuentra siempre en memoria, mientras que un archivo real se encuentra en el disco duro u otro dispositivo de almacenamiento.

Existen también dos constructores, el primero es para manipular archivos de solo lectura, el cual recibe como único parámetro el nombre del archivo, e internamente solo ejecuta la llamada al sistema `open()`. El segundo constructor es para archivos de escritura y recibe adicionalmente las banderas y el modo del archivo (véase la práctica 8).

El método `lee` realiza una lectura de n bytes sobre el archivo en disco duro, mediante la utilización de la llamada al sistema `read`; después almacena la información en la variable privada `contenido`, previa reserva de memoria con ayuda de la función `realloc`, y devuelve el número de bytes que se lograron leer. Si se le vuelve a mandar llamar continuará leyendo el archivo donde se había quedado y entonces será necesario reasignar más memoria con `realloc` para que el objeto contenga los siguientes bytes leídos. Esto explica porque no es posible solo utilizar `malloc`. Por otro lado se almacena como caracteres del tipo C porque usamos llamadas al

sistema UNIX que así lo requieren, y porque un archivo no necesariamente es de texto, también puede ser un archivo de video o música.

El método `escribe` realiza una escritura de `nbytes` bytes en el archivo mediante la utilización de la llamada al sistema `write`. Los bytes a escribir deberán ser proporcionados mediante un apuntador al área de memoria externa en el primer parámetro. El método debe devolver el número de bytes que se lograron escribir en el archivo.

El método `obtieneNum_bytes` devuelve el número de bytes que se encuentran almacenados dentro del objeto hasta el momento.

El método `get_contenido` devuelve un apuntador a la variable contenido. Observe que aunque la variable es privada, si se obtiene su apuntador, podría modificarse su contenido desde el programa principal, por esa razón se antepone la palabra reservada `const`. Pruebe esta última aseveración.

El destructor debe liberar la memoria que se haya reservado además de cerrar el archivo mediante la función `close`.

Ejercicio 1: Para probar su clase Archivo realice la misma operación que el programa 8-1 del manual pero con el código orientado a objetos en C++.