Ampliación de Sistemas Operativos

Práctica 2. Sincronización de directorios usando inotify.

Funcionamiento general

Se pide el desarrollo de un programa `sincro.x`, que mantenga sincronizados dos directorios (en adelante, `dir_origen` y `dir_destino`). El programa leerá un fichero de configuración llamda `.sincro.conf` que residirá en el directorio `HOME` del usuario.

En dicho fichero (que se creará a mano antes de la ejecución del código) deberá haber dos líneas de texto indicando el `dir_origen` y el `dir_destino` respectivamente. Si el directorio origen no existe, el programa reportará al usuario un error y finalizará. Si el directorio destino no existe, el programa lo creará (llamada al sistema `mkdir`).

Funcionalidad básica (hasta 7 puntos).

Dados dos directorios `dir_origen` y `dir_destino`, se considerarán sincronizados a partir de un instante de tiempo, si para cada **fichero regular** que se cree/modifique/elimine del primero, se realiza la misma acción en el destino (almacenando idénticos datos, permisos e información de propietario y grupo propietario)

Nótese que no esto no implica que ambos directorios sean idénticos: se mantendrán sincronizados a partir de la ejecución de `sincro.x`, pero si el contenido original de ambos directorios era diferente, esa parte podrá seguir siendo diferente [1]

Para simplificar el proceso, se usará `inotify()` sobre el directorio origen, de manera que la aplicación reciba notificaciones cuando se produzca cualquier cambio en los ficheros regulares que contenga. Así, para cada notificación recibida por algún cambio en un fichero en `dir_origen`:

- Si el fichero no existe en `dir_destino`, lo creará (llamada al sistema `open` o `creat`), copiará los datos que formen parte de él y establecerá correctamente información sobre permisos y propietarios.
- Si el fichero ya existe en `dir_destino`, se trata de una modificación. Entonces el programa actualizará el contenido del fichero en `dir_destino` para mantenerlo sincronizado con el origen.
- Si el fichero en `dir_origen` y el fichero en `dir_destino` difieren en su información sobre permisos o propietario, el programa los mantendrá consistentes (llamadas al sistema `chmod` y `chown`).
- Si la notificación es por la eliminación de un fichero, se eliminará de `dir_destino` si existía allí.
- La función de copia entre ficheros se implementará en un fichero aparte (`copia.c`, con su correspondiente fichero de cabecera `copia.h`) y tendrá como prototipo `int copia(int fdo, int fdd)`, siendo `fdo` el descriptor del fichero origen y `fdd` el descriptor de fichero destino. Devolverá un entero que indicará el número de bytes copiados o *-1* en caso de error. Para esta primera versión básica, la copia de datos se realizará byte a byte, utilizando llamadas al sistema `read` y `write`.

[1] Para probar la práctica, es recomendable comenzar con dos directorios vacíos. Así, realmente el contenido de ambos deberá ser idéntico

Se recomienda encarecidamente leer el documento *Monitor Linux system events with inotify* (https://www.ibm.com/developerworks/library/l-inotify/l-inotify-pdf.pdf). Además de describir en detalle el uso de `inotify()`, proporciona un **excelente código de partida** para la práctica.

Ampliación. Uso de poll() (hasta 2 puntos adicional).

Si se ha usado el código del artículo de IBM referenciado en el apartado anterior, se habrá notado el uso de `select()` para realizar la espera de notificaciones.

En este apartado se pide cambiar el uso de `select()` por el de la función similar `poll()` de modo que la aplicación funcione exactamente igual.

Ampliación. Carga dinámica de función de copia (hasta 3 puntos adicionales).

Para esta ampliación se deberá usar la función `dlopen()` que permite la carga dinámica de librerías. Se incluirá una tercera línea en el fichero de configuración (`.sincro.conf`) indicando el nombre de la librería dinámica (*.so*) que deberá cargarse con `dlopen()` [2].

Como primera prueba, se compilará el fichero `copia.c` implementado en la parte básica como una librería dinámica (`libcopiaByte.so`) y se cargará de forma dinámica.

Posteriormente, se implementará una función `copia` alternativa (con el mismo prototipo) en un nuevo fichero (`copiammap.c`). Esta copia se implementará usando `mmap()` para mapear los ficheros en memoria antes de la copia. Nuevamente, se compilará como una librería dinámica y se comprobará el funcionamiento correcto de la aplicación `sincro.x`.

Notas.

- No se tendrán en cuenta los enlaces duros en el proceso de sincronización (se tratarán como ficheros regulares sin enlaces a ellos).
- Los directorios a sincronizar serán totalmente independientes en cualquier caso (esto es, el directorio destino no será un subdirectorio de ninguno de los subdirectorios del directorio origen).
- Para el manejo de cadenas, se recomienda el uso de rutinas de tipo `strncat` (para concatenar dos cadenas) y `strncpy` (para realizar la copia de cadenas).
- No se permite el uso de llamadas al sistema de tipo `system` o cualquiera de la familia `exec`.
- Se valorará el control de errores en todas las llamadas al sistema utilizadas.

[2] Busca información sobre `dlopen()` en internet para aprender a utilizarlo. Una muestra en https://www.dwheeler.com/program-library/Program-Library-HOWTO/x172.html