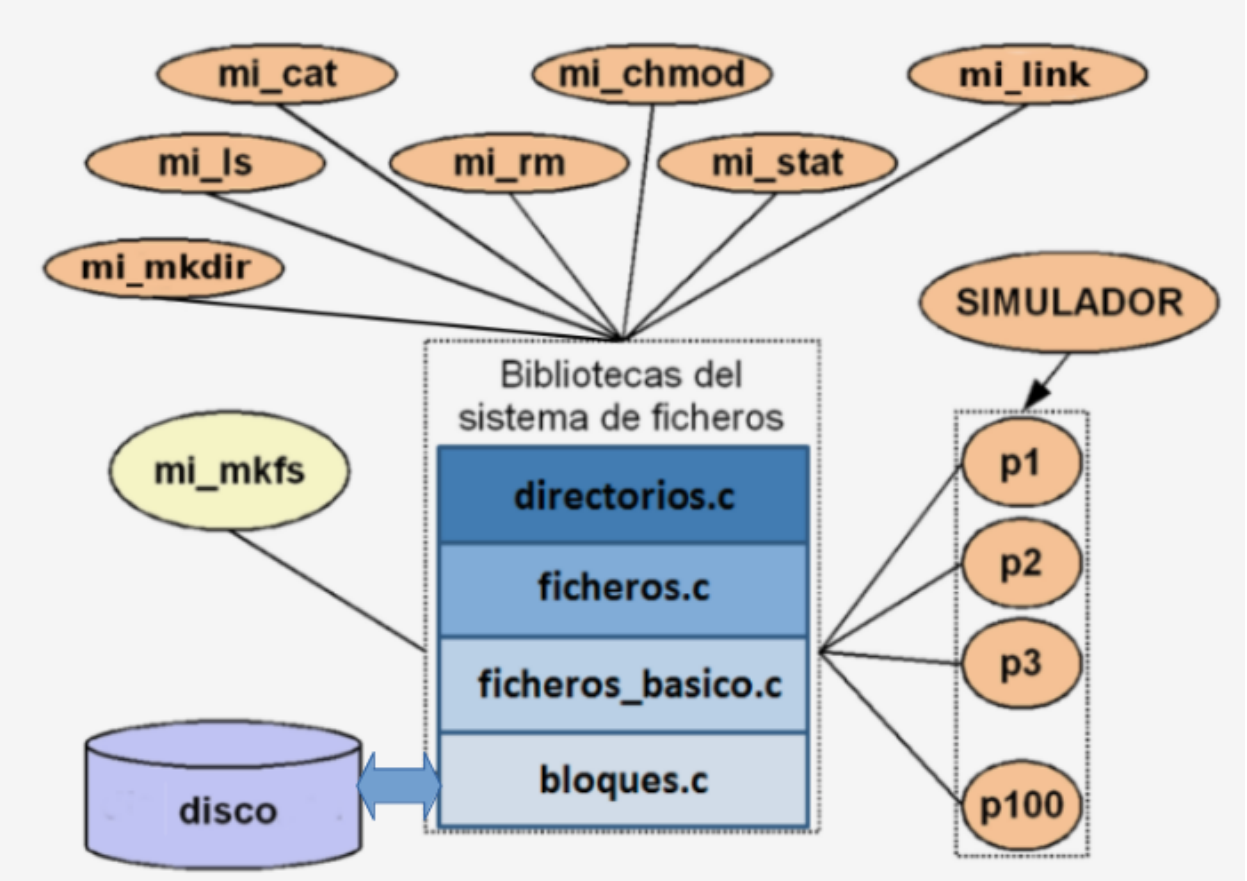
Sistema de ficheros

Enunciado general

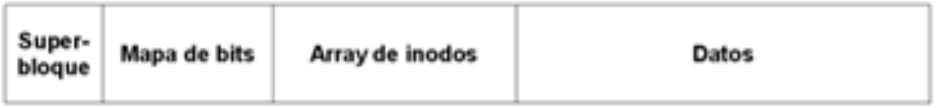
La práctica implementará un sistema de ficheros (basado en el ext2 de Unix) en un disco virtual (fichero), el módulo de gestión de ficheros del sistema operativo y la interacción de procesos de usuarios.



Estructura general

La práctica estará compuesta por los siguientes elementos:

- Un conjunto de **bibliotecas**, divididas en niveles, que darán la funcionalidad a las primitivas para acceder al sistema de ficheros y gestionarlo.
- Un programa **mi_mkfs** para crear, con la ayuda de dichas bibliotecas, el sistema de ficheros en el disco.
- El sistema de ficheros (SF) estará realmente contenido en un **fichero** (simulando un dispositivo de memoria secundaria), al que llamaremos **disco** y tiene la siguiente estructura:



- Un conjunto de programas para operar por consola, entre ellos uno que muestre el contenido de un fichero por pantalla (***mi_cat***), uno que cambie sus permisos de acceso (***mi_chmod***), uno uno que cree enlaces físicos (***mi_ln***), uno que liste directorios (***mi_ls***), uno que borre ficheros y directorios (***mi_rm***), y uno que muestre la metainformación de un fichero (***mi_stat***).
  - Su funcionamiento es similar al funcionamiento básico de los correspondientes programas *cat*, *chmod*, *ln*, *ls*, *rm*/*rmdir*, *stat* de GNU/Linux.
- Un programa **simulador** encargado de crear unos **procesos** de prueba que accedan de forma concurrente al sistema de ficheros (utilizando un semáforo mutex) de modo que se pueda comprobar el correcto funcionamiento de las citadas bibliotecas.

Requisitos

Los requisitos mínimos son:

- El sistema de ficheros deberá ser jerárquico, con estructura de árbol de directorios y ficheros, pero también se podrán crear enlaces físicos, lo cual en realidad lo convertirá en un grafo.
- El sistema de ficheros debe ser de **asignación indexada** (inodos de 128 bytes con 12 punteros directos, 1 puntero indirecto simple, 1 puntero indirecto doble y 1 puntero indirecto triple).
- Cada fichero (es decir, cada inodo) tendrá una metainformación mínima necesaria para las primitivas.
- El sistema de ficheros debe estar preparado para trabajar con un tamaño de bloque comprendido entre 512 bytes y 4.096 bytes (ha de ser una constante en el programa para poder cambiar fácilmente el valor si se desea)
- El nombre del fichero que hace de dispositivo virtual y que almacenará el sistema de ficheros deberá ser dinámico; es decir, no estará definido en los programas fuentes.
- La asignación de espacio a los ficheros deberá ser dinámica: se asignarán nuevos bloques a medida que hagan falta.
- En las bibliotecas deben implementarse métodos de control de concurrencia para acceder a los metadatos de gestión del sistema de ficheros

(superbloque, mapa de bits y array de inodos): hay que definir secciones críticas y usar semáforos donde corresponda.

- Dentro de las bibliotecas no se ha de controlar el acceso concurrente de varios procesos a los datos de un mismo fichero, ya que eso es responsabilidad de las aplicaciones.

Primitivas de acceso

Las siguientes operaciones son las **mínimas** que deben estar programadas en las bibliotecas (directorios.c) del sistema de ficheros:

mi_dir(): Obtiene el contenido de un directorio en una cadena de caracteres (puede separar su contenido por ':') así como la cantidad de ficheros listados.

mi_stat(): Obtiene la metainformación de un fichero (es decir, la de su inodo).

mi_read(): Lee una cantidad determinada de bytes de un fichero, desde una posición inicial, los pone en un *buffer* de memoria y devuelve la cantidad total de bytes leídos.

mi_write(): Escribe en un fichero, desde una posición inicial, una cantidad determinada de bytes a partir de un *buffer* de memoria y devuelve la cantidad total de bytes escritos.

mi_creat(): Crea el fichero/directorio especificado.

mi_link(): Crea el enlace de una entrada de directorio al inodo especificado por otra entrada de directorio. Ambas entradas se han de corresponder a ficheros.

mi_unlink(): Borra el enlace físico especificado y, en caso de que sea el último existente, borra el propio fichero.

mi_chmod(): Cambia los permisos de acceso al fichero.

Además se han de gestionar los posibles errores devueltos por las funciones.

Simulación

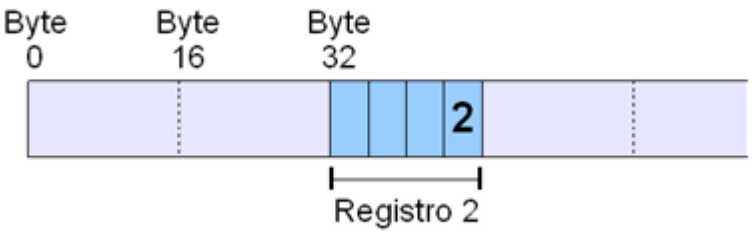
La simulación consta de tres partes.

1) Escrituras concurrentes en el sistema de ficheros

Antes de comenzar, se creará el directorio "`simul_aaaammddhhmmss`", donde *aaaa* es el año, *mm* es el mes, *dd* es el día, *hh* es la hora, *mm* es el minuto y *ss* es el segundo de creación.
Se han de generar 100 procesos de prueba[1] cada 0,2 segundos[2]. Cada proceso creará un directorio llamado "`proceso_n`", donde n es el PID del proceso, dentro del directorio "`simul_aaaammddhhmmss`".
Además, dentro del directorio "`proceso_n`", cada proceso creará un fichero "`prueba.dat`".
Cada 0,05 segundos, y un total de 50 veces, cada proceso escribirá dentro del fichero "`prueba.dat`" un registro con la siguiente estructura de datos:

- Entero (*time_t*) con la fecha y hora en forma *epoch*
- Entero con el PID del proceso que lo creó
- Entero con el número de escritura (de 1 a 50)
- Entero con la posición del registro dentro del fichero (**número de registro**)

Esta posición se elegirá de manera aleatoria, teniendo en cuenta que el registro puede ocupar cualquier ubicación dentro del espacio de datos del fichero. De esta manera comprobaremos el correcto funcionamiento de los punteros directos e indirectos de los inodos.
No importa si algún registro sobreescribe alguno de los registros anteriores.



2) Verificación de las escrituras

Cuando acaben las escrituras, se verificará el fichero "`prueba.dat`" de cada proceso y se generará la siguiente información:

- **Proceso**: escribir el PID

[1] La función fork() permite crear un nuevo proceso
[2] Se puede provocar esa espera mediante la función usleep

- **Número de escrituras**: escribir el contador de los registros *validados* dentro del fichero "`prueba.dat`" (se validan verificando que el campo PID coincida con el especificado en el nombre del directorio)
- **Primera escritura**: escribir la menor fecha y hora en formato *epoch*, el número de escritura en que ocurrió y su posición
- **Última escritura**: escribir la mayor fecha y hora en forma *epoch*, el número de escritura en que ocurrió y su posición
- **Menor posición**: escribir la posición más baja, el número de escritura en que ocurrió y su fecha y hora en forma *epoch*
- **Mayor posición**: escribir la posición más alta, el número de escritura en que ocurrió y su fecha y hora en forma *epoch*

El objetivo es imprimir por pantalla los cuatro registros más significativos que cada proceso ha escrito en su fichero.
El número de escrituras, por regla general, tiene que valer 50.

3) Pruebas con *mi_cat*, *mi_chmod*, *mi_ln*, *mi_ls*, *mi_rm*, *mi_stat*, ...

Por último, con la ayuda de la infraestructura recién creada, diseñar un *shell script* (o varios *shell scripts*) para mostrar el correcto funcionamiento de los programas ***mi_cat***, ***mi_chmod***, ***mi_ln***, ***mi_ls***, ***mi_rm***, ***mi_stat*** desarrollados para llamar a las correspondientes primitivas.