Universidad Tecnológica Nacional – Facultad Regional Buenos Aires
Ingeniería en Sistemas de Información
Sistemas Operativos (082027)

Trabajo Práctico

Sistemas Operativos

FUSELAGE

(**FUSE** *based*, **L**ayered, **A**sinchronic, **G**eographically distribut**E**d, *fat32 file system*)

2do cuatrimestre 2011

v1.0

Índice

1.- Introducción

El trabajo práctico de este cuatrimestre consiste en la implementación de un sistema de archivos que opere volúmenes con el formato FAT32 sobre un sistema operativo GNU/Linux. Dicha implementación trabajará con un dispositivo de almacenamiento secundario distribuido en otro sistema, el cual implementará un esquema simplificado de RAID 1 sobre un procesamiento planificado de los pedidos al disco.

El objetivo del mismo es que mediante el desarrollo de esta especificación el alumno se interiorice y comprenda los diversos mecanismos que utiliza el sistema operativo para gestionar los medios de almacenamiento y su relación e interacción con los sistemas de archivos.

No se pretende enfocar en el funcionamiento específico de un determinado sistema operativo sino por el contrario mostrar conceptos y aspectos de diseño generales.

Dada la complejidad del mismo, este trabajo fue dividido en **checkpoints** los cuales permiten diseñar la arquitectura definitiva, de forma gradual, simplificando su desarrollo. Esta parte se encuentra mas detallada en el apartado **Ciclo de Desarrollo**.

Se enumeran a continuación los conceptos teóricos y prácticos más significativos sobre sistemas operativos que cubre el trabajo práctico y que el alumno aprenderá:

- Procesos
  - Creación
  - IPC
    - Sockets TCP
    - Sockets Unix
  - Manejo de Señales
  - Manejo de hilos mediante la API del sistema
    - Creación
    - Sincronización

- Memoria
  - Administración de memoria dinámica

- Entrada/Salida
  - Funcionamiento de RAID1

- Sistema de Archivos
  - Implementacion de FAT32

- Diseño de SO
  - Arquitecturas basadas en capas físicas y lógicas

- Protocolos

2.- Objetivos del Trabajo Práctico

Desde el punto de vista académico el trabajo está diseñado para que el alumno:

- Adquiera los conocimientos prácticos del uso y aplicación de un conjunto de servicios que ofrecen los sistemas operativos modernos.
- Domine los problemas específicos de este tipo de implementaciones.
- Evalúe las ventajas y desventajas de la utilización de soluciones similares a un mismo problema.
- Afronte problemas de diseño en los componentes propios de un sistema operativo.
- Entienda la importancia de una norma o protocolo estándar en la comunicación entre procesos y diferentes plataformas.
- Se entrene en el trabajo en equipo, manejo de las problemáticas de un grupo y las responsabilidades que esto implica.

Este trabajo práctico ***no es una especificación completa***, sino que brinda los lineamientos principales de los objetivos a cumplir. Cualquier detalle que en el trabajo práctico no esté especificado y necesite ser definido para el desarrollo del mismo, deberá ser consultado y si no resulta ser un error de enunciado, *el grupo deberá tomar sus propias decisiones de implementación, las cuales deberán ser documentadas*.

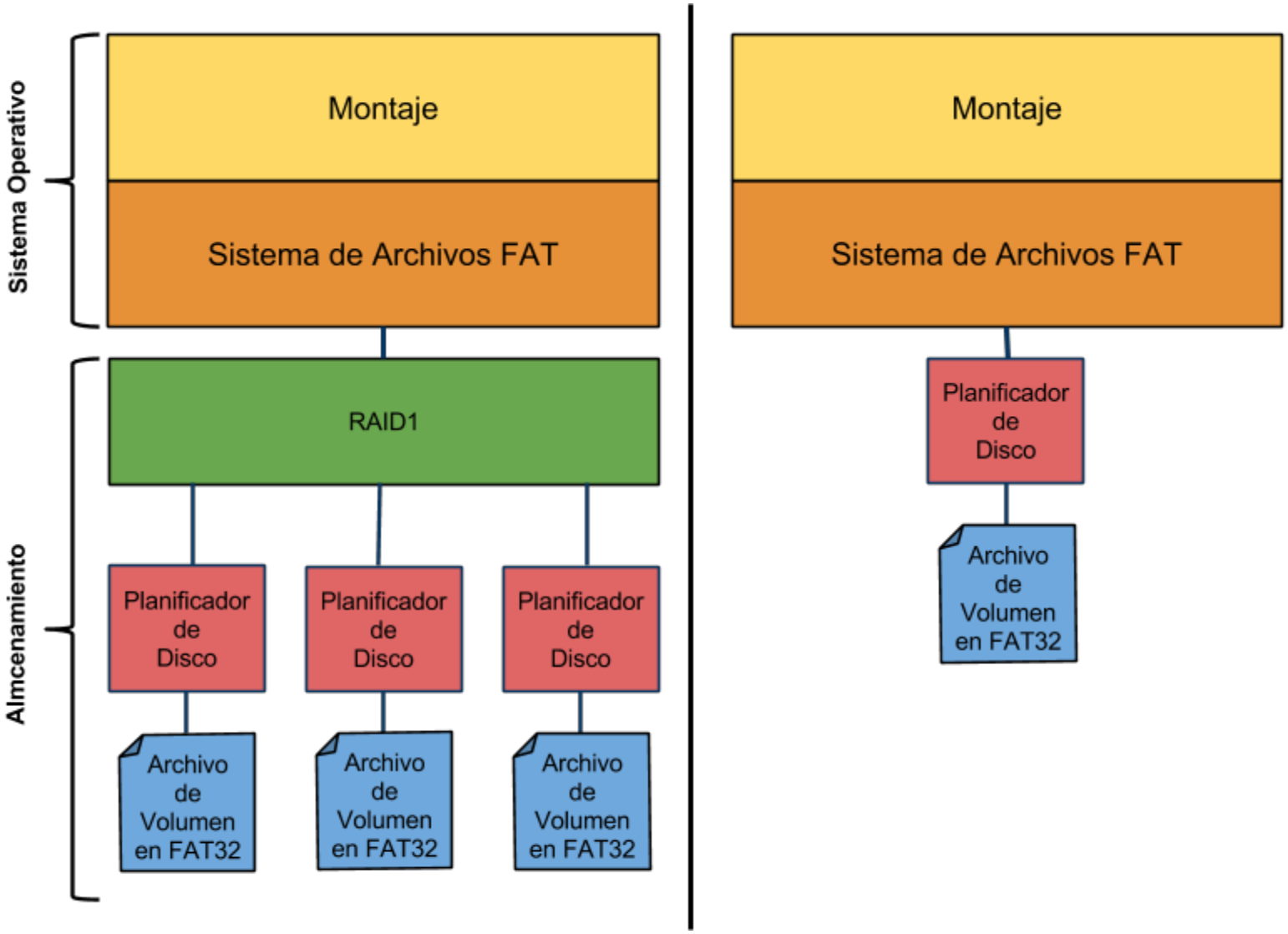
La idea de tener una especificación más abierta es que los grupos aprendan a analizar diferentes alternativas para resolver un mismo problema y, al mismo tiempo, evaluar los costos y beneficios de implementar cada una de ellas.

Finalmente en la especificación de cada proceso, se mencionará un archivo de configuración en el cual se detallaran campos obligatorios para lograr las funcionalidades especificadas. Pero el grupo puede sentirse libre de agregar los campos extra que necesite, que sean requeridos por su implementación.

3.- Características del Sistema

Diagrama de componentes

El sistema está conformado por dos capas, implementadas mediante distintos procesos, con responsabilidades claramente definidas. Estas capas son: “Sistema Operativo” y “Almacenamiento”. A continuación se presenta un diagrama de arquitectura general del sistema:



Sistema Operativo

Esta capa contiene los componentes que interactuarán con el kernel del sistema operativo anfitrión para brindar los servicios y las funcionalidades necesarias para operar con el sistema de archivos FAT32. Estará conformada por el proceso denominado “Proceso File System” el cual estará compuesto por 2 subcapas:

- **Montaje:** Consistirá en un mecanismo para brindar al kernel las operaciones a realizar por nuestro sistema de archivos FAT32. Es decir, este componente actuará como un puente entre nuestra implementación de FAT32 y el Kernel.

- **Sistema de Archivos FAT:** Estará encargada de interpretar el sistema de archivos FAT32 y brindar al sistema operativo anfitrión, a través de la capa de montaje, las funcionalidades necesarias como para poder montar el filesystem e interactuar con éste.

  Ambas subcapas se encontrarán dentro de un mismo un mismo proceso, el cual montará el sistema de archivos FAT del disco en un directorio.

Almacenamiento

Esta capa estará compuesta por los dispositivos que facilitan la persistencia de la información.

- **Proceso RAID1:** Este proceso implementará un esquema RAID nivel 1, al cual se le conectarán procesos discos y este se encargará de mantener la información redundada entre todos ellos. Por simplicidad, dicho esquema de RAID1 contará con un solo “dispositivo de datos”, siendo el resto de los mismos réplicas iguales de dichos datos. Para los procesos de la capa de Sistema Operativo, interactuar con RAID1 o directamente con un disco es transparente.

- **Proceso Planificador del Disco:** Este proceso es el que planifica los pedidos de lectura y escritura, y realiza las operaciones sobre el ***Archivo de Volumen en FAT32***.

- **Archivo de Volumen FAT32:** Este componente es simplemente un archivo el cual contiene los datos de un disco formateado con el filesystem FAT32. Lo que le permite simular ser un disco sobre el cual va a interactuar el Planificador Del Disco.

Cache

Este componente, se encuentra presente dentro de la ***capa*** del Sistema Operativo. Existe la posibilidad de que por cuestiones de diseño, simplicidad o performance el alumno considere necesario agregar otros niveles de caché. En estos casos su comportamiento y ubicación dentro del sistema quedará a discreción del grupo. La forma en que lo implementen será parte de la evaluación del TP.

4.- Aspectos de diseño de los procesos

Proceso File System (PFS)

Este proceso es el encargado de leer un volumen FAT32, accediendo a este a través del Proceso Planificador del Disco ó del Proceso RAID 1, y exponer las operaciones que se pueden realizar sobre el mismo al kernel de sistema operativo anfitrión.

Al ejecutar nuestro proceso, este recibe por parámetro como mínimo el path (ruta) del directorio donde vamos a montar nuestro File System FAT32. A partir de esto comienza el funcionamiento de nuestro proceso, el cual funciona de manera concurrente y asincrónica.

Como todo File System, el nuestro también trabaja con un dispositivo de almacenamiento, solo que éste es simulado por los procesos de la capa de almacenamiento. Al igual que estos, solo saben manipular **sectores**, ya que esta es la mínima unidad que puede manejar los dispositivos. Por el contrario, los File Systems no trabajan directamente sobre los sectores sino que trabajan sobre el concepto de **bloque**.

Un bloque es la mínima unidad que pueden manejar internamente los File System y que su tamaño, en el caso de los sistemas Linux, puede variar entre 1 y **n** sectores pero nunca mas grande que el tamaño de una pagina de memoria. Para el caso de este TP, el grupo deberá elegir el valor de **n** siendo este siempre mayor a 1.

La subcapa FAT trabaja solicitando o enviando bloques, los cuales, cada uno, constan de **n** pedidos a nivel sectores, al Proceso Planificador del Disco o al Proceso RAID. Las operaciones tanto para un proceso como para el otro son las mismas por lo que para nuestro Proceso File System estar conectado a uno o a el otro es completamente transparente. El protocolo de conexión entre estos procesos se encuentra detallado en el **Anexo B**.

El objetivo de este proceso es que pueda brindar las siguientes operaciones al usuarios:

- Crear un archivo
- Abrir un archivo
- Leer un archivo
- Escribir un archivo
- Borrar un archivo
- Truncar un archivo
- Flush de un archivo
- Cerrar un archivo
- Crear un directorio
- Leer un directorio
- Borrar un directorio
- Obtener los atributos de un archivo/directorio
- Renombrar un archivo/directorio

Proceso RAID 1 (PRAID)

Este proceso, implemente un esquema RAID 1, el cual proporciona redundancia mediante la simple estrategia de duplicar todos los datos. Por cuestiones de simplificación, dicho esquema tendrá un solo disco de datos, siendo el resto discos de redundancia.

Los aspectos básicos del funcionamiento esperado:

1. Una petición de lectura debe poder utilizar cualquiera de los discos, quedando a criterio del grupo el seleccionar o diseñar un algoritmo de distribución de peticiones de lectura. Esto mismo será evaluado en el coloquio.
2. Una petición de escritura requiere actualizar todos los discos, es por ello que ésta se debe realizar en paralelo.

Proceso Planificador de Disco (PPD)

Este proceso implementará un planificador de peticiones a nuestro “disco”, sin interleaving, teniendo un funcionamiento completamente asincrónico. Tendrá como características sus valores de CHS, RPM, velocidad de lectura de un sector y posición actual del cabezal de lectura, los cuales serán provistos inicialmente por archivo de configuración. Por simplicidad de implementación **el disco contará con un solo cabezal.**

El planificador, al recibir una solicitud sobre un sector, deberá calcular el tiempo que le insumiría alcanzar dicho sector. Hay que tener en cuenta que al ser una simulación y con el objetivo de no aumentar la complejidad del TP, durante el tiempo que se finaliza un pedido y se espera a que llegue otro el disco no tiene movimiento alguno.

5.- Aspectos Técnicos de los Procesos

Proceso File System (PFS)

Para exponer funcionalidades de un File System al Kernel de Linux los alumnos deberán utilizar la biblioteca FUSE (http://fuse.sourceforge.net/):

FUSE (Filesystem in Userspace)
Es un módulo de Kernel para sistemas operativos tipo Unix, que permite a usuarios no privilegiados crear sus propios sistemas de archivos sin necesidad de editar el código del kernel. Ésto se logra mediante la ejecución del código del sistema de archivos en el espacio de usuario, mientras que el módulo FUSE sólo proporciona un "puente" a la interfaz del núcleo real.

Las operaciones que tienen que ser implementadas en FUSE son las correspondientes a las enumeradas previamente, a saber:

- *create* /** Create and open a file */
- *open* /** File open operation */
- *read* /** Read data from an open file */
- *write* /** Write data to an open file */
- *flush* /** Possibly flush cached data **/
- *release* /** Release an open file **/
- *truncate* o *ftruncate* /** Changes the size of file **/
- *unlink* /** Remove a file */
- *mkdir* /** Create a directory */
- *readdir* /** Read directory */
- *rmdir* /** Remove a directory */
- *fgetattr* o *getattr* /** Get file attributes. */
- *rename* /** Rename a file */

Nuestro File System debe ser capaz de interactuar con un volumen FAT32. Para ello el grupo deberá usar, **y podrá extender**, el protocolo NIPC especificado en el **Anexo B** para comunicarse con el Proceso Planificador del Disco o el Proceso RAID1 y así interactuar con el almacenamiento de los datos. El Proceso Planificador del Disco leerá/escribirá en un volumen, **el cual ya se encontrará formateado en FAT32** y que podría tener archivos creados en su interior.

Con la información y documentación provista en el **Anexo C** el grupo tendrá una especificación para implementar FAT32 para poder interpretar la información que brindará el Proceso Planificador del Disco.

Concurrencia

Como todo File System, este recibe peticiones concurrentemente. Es decir, se puede estar abriendo un archivo mientras se esta leyendo otro. Incluso un mismo archivo puede estar siendo accedido al mismo tiempo desde varias aplicaciones diferentes. Durante la evaluación, FUSE funcionará en modo multithread[1], dando la posibilidad de que se realicen operaciones en simultáneo y es por esto que existirá más de una conexión hacia el RAID/Planificador. El objetivo de las múltiples conexiones es tratar de maximizar el nivel de paralelismo de las operaciones y por ende el de las aplicaciones, ya que si utilizáramos solo una conexión nos conduciría a reducir drásticamente el desempeño del sistema, a menos que se diseñen mecanismos más complejos de sincronización.

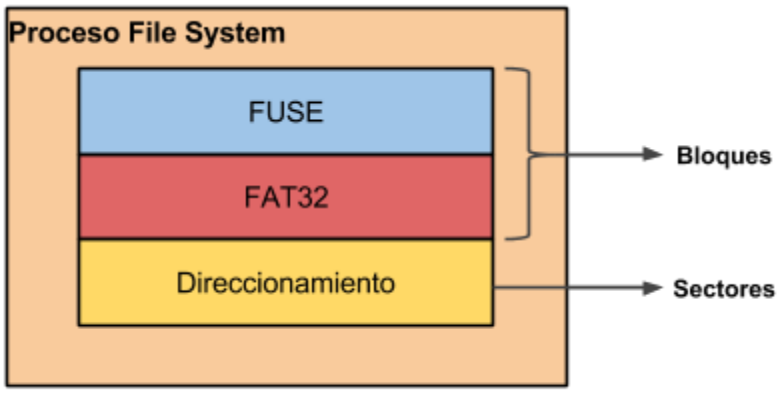
Cuando FUSE solicite una operación que requiere del acceso al dispositivo de almacenamiento, se debe optar por:

- Crear una nueva conexión hacia el RAID/Planificador y cerrarla al finalizar la operación.
- Crear un pool (conjunto) de conexiones al comienzo del proceso y utilizar una conexión que se encuentre libre.

En cualquiera de los casos existirá una cantidad máxima de conexiones establecidas simultáneamente, las cuales estarán estipuladas en el archivo de configuración.

Asincronismo

Como se detalló anteriormente, el File System trabaja a nivel de bloques, los cuales están compuestos por **n** sectores. Por ello el File System requiere de una capa de direccionamiento con el que convierte el pedido de un bloque dado, en **n** pedidos asincrónicos de sectores.



Capas y Unidad de Operación

[1] *Durante el desarrollo del trabajo practico, y solamente por propósitos de testeo y debugging, los alumnos pueden optar por correr FUSE en modo single-thread.*

Cache

Para tratar de maximizar el rendimiento en la escritura/lectura, el grupo deberá implementar una cache de lectura/escritura dentro del Proceso File System, por cada archivo con el que se esté operando.
El tamaño de esta cache estará definido en el archivo de configuración, siendo 0 la cache deshabilitada o un valor distinto de 0 que sea exponente de 2 entre 4kb y 32kb (este tamaño corresponde exclusivamente a los datos que la misma almacene, descartando cualquier camp extra de tipo administrativo).

Durante la ejecución del Proceso File System será posible enviar a este la señal SIGUSR1. Este creará, como consecuencia de la misma, un archivo llamado "*cache_dump.txt*"[2], el cual contendrá un snapshot del estado de la cache de cada archivo, en un formato legible. Este mismo comando será utilizado durante las pruebas para corroborar el funcionamiento de la cache y su contenido.

Ej. de Formato:

Timestamp: 2011.08.03 14:00:00
Archivo: pepe.txt
Tamaño de Bloque de Cache: 4kb
Cantidad de Bloques de Cache: 2

Contenido de Bloque de Cache 1:
Esto es una prueba.

Contenido de Bloque de Cache 2:
….

Timestamp: 2011.08.03 14:00:00
Archivo: pepe.txt
Tamaño de Bloque de Cache: 4kb
Cantidad de Bloques de Cache: 2

Contenido de Bloque de Cache 1:
Esto es una prueba.

Contenido de Bloque de Cache 2:
….

[2] *En caso de que el archivo ya exista, se debe utilizar el existente y comenzar a escribir luego de la información que ya se encuentre en este.*

Cuando se muestra el contenido de un bloque, se debe mostrar todo el contenido, por más que existan '\0' en el medio de la información.

Consola

El Proceso File System poseerá una consola como medio de entrada, a través del stdin, por el cual se le podrán ingresar los siguientes comandos:

- ***fsinfo*** (File System Information): imprimirá en pantalla la cantidad de clusters ocupados, la cantidad de clusters libres, el tamaño de un sector *(en bytes)*, el de un cluster *(en bytes)* y el tamaño de la FAT *(en kilobytes)*.

- ***finfo [path a un archivo]*** (File Information): Dado el path a un archivo, se listará (en orden en el que esté dispuesto el contenido del archivo) como máximo los 20 primeros números de clusters asociados a un archivo.

  Ej:
  finfo /test/algo.txt / Compuesto por 11 clusters */*

  #Clusters:
  20, 21, 19, 56, 57, 58, 62, 10, 64, 65, 66.

Notas:

- La función *getattr* permite manipular los valores del struct stat del archivo. Será responsabilidad del filesystem implementado setearlo de la manera adecuada. Ej: Cantidad de hard links, permisos de escritura/lectura para los archivos, etc …

- No será necesario manejar la ultima fecha de acceso y modificación del archivo.

Datos de Configuración

- IP y Puerto de conexión al RAID/Planificador
- Máxima cantidad de conexiones
- Tamaño de la Cache

Proceso RAID 1 (PRAID)

Este proceso debe exponer una interfaz de comunicación, a través el protocolo NIPC especificado en el **Anexo B**, con el Proceso FileSystem tal que sea transparente para este, tanto si se trata de un RAID o de un Planificador del Disco. Al ejecutarse, este se pondrá a la escucha de nuevas conexiones por parte de Procesos Planificadores de Discos. Cuando uno o más Planificadores del Discos se conectan al Proceso RAID, este entrará en funcionamiento por lo que recién en esta instancia el Proceso FileSystem puede conectarse al Proceso RAID (En caso contrario, durante el handshake entre el Proceso FileSystem y el Proceso RAID, este último deberá rechazarlo e informar que aún no esta activo).

A su vez el Proceso RAID funcionará completamente de forma asincrónica, es decir, a medida que las peticiones lleguen se irán procesando y cuando los discos respondan se enviarán estas respuestas al Proceso File System.

Cuando un Proceso Planificador del Disco se conecte al Proceso RAID este realizará los siguientes pasos:

- Al nuevo proceso Planificador del Disco se le asignará un thread.

- El nuevo disco deberá ser sincronizado, para contener la misma información que el resto de los discos.

Cabe destacar que el RAID no detuvo su ejecución durante estos pasos, sino que el Proceso RAID 1 debe seguir atendiendo pedidos siempre y cuando estos no afecten la sincronización de datos con el nuevo disco generando inconsistencias. De lo contrario, los pedidos quedaran encolados a la espera de que su ejecución no afecte la sincronización.

Si un Proceso Planificador del Disco se cae, el Proceso RAID 1 deberá seguir operando con completa normalidad. Los pedidos que aún se encontraban pendientes para ese Proceso Planificador del Disco caído deben ser replanificados entre el resto de los discos funcionales. En caso de que el Proceso Planificador del Disco que se caiga sea el disco a partir del cual se estén replicando los datos, el sistema debe finalizar su ejecución.

Consola

El Proceso RAID posee una consola la cual únicamente se encarga de informar:

- Cuando el Proceso RAID entra en funcionamiento
- Cuando se conecta un nuevo Planificador del Disco al RAID y el identificador de este
- Cuando se sincronizan los discos [Tiempo de inicio y fin de esta tarea]
- Cuando se hace un write en todos los discos
- Cuando se hace un read y de que disco se esta leyendo

Datos de Configuración

- Puerto TCP de escucha tanto para el Proceso File System como para Proceso Disco
- Flag para activar o desactivar la consola.

Proceso Planificador de Disco (PPD)

Tal y como ocurre con el Proceso RAID, el Planificador de Disco, proveerá una API definida por el alumno con la cual se podrá comunicar con este o con el Proceso File System de forma completamente transparente utilizando el protocolo NIPC especificado en el **Anexo B**. El Planificador de Disco debe proveer 2 funcionalidades básicas para trabajar con sectores (Los cuales siempre tendrán un tamaño de 512 bytes):

- Leer Sector: El disco al recibir un número de sector deberá colocarlo en la cola de pedidos. Para facilitar las pruebas **se debe poder configurar un retardo en segundos a la operación** de lectura parametrizable por archivo de configuración. Una vez leído, el sector es devuelto al Proceso RAID para que luego este lo devuelva al Proceso FileSystem ó al Proceso File System en caso de que estos estén conectados directamente.

- Escribir Sector: El disco recibe un número de sector junto con su contenido de 512 bytes lo cual es encolado para posteriomente ser atendido.
  Para facilitar las pruebas **se debe poder configurar un retardo en segundos a la operación** de escritura, parametrizable por archivo de configuración (distinto al valor utilizado para lectura).

El funcionamiento es completamente asincrónico, es decir, ante la llegada de una petición, la misma no deberá ser tratada inmediatamente, sino que será encolada y tratada cuando sea oportuno de acuerdo al algoritmo de planificación usado. Un vez seleccionada y atendida la petición, esta sera respondida al Proceso RAID o al Proceso File System.

El manejo de pedidos de escritura y lectura pueden ser tratados de diferente manera, ya sea para definir si se manejan varias colas, escritura bajo demanda (siempre que no se supere el tamaño de la cola), establecer que la lectura tiene mas prioridad sobre la escritura, etc. Ésto debe ser analizado, diseñado, implementado y documentado por el grupo.
El medio sobre el cual el Proceso Planificador de Disco realizará la lectura y/o escritura, no será una partición de un disco rígido real. A fines prácticos el Proceso Planificador de Disco trabajará sobre un archivo formateado bajo FAT32. Este archivo puede ser creado mediante el comando **mkfs.msdos**.

Este comando es capaz de crear un archivo formateado como FAT32, como si se tratara de una partición de un disco, especificándole parámetros como tamaño de sector, cantidad de sectores por cluster, etc. Por ejemplo:

```
$ mkfs.msdos -F 32 -S 512 -s 8 -C fat32.disk 524288
```

Para probar la integridad del archivo y su contenido se puede utilizar el comando mount, para montar el archivo como si se tratase de unidad de disco físico, utilizando el filesystem *nativo* de FAT32 provisto por el sistema operativo anfitrión:

```
$ mount -o loop <path de nuestro archivo> <punto de montaje>
```

El método para operar (léase: escribir/leer) sobre el archivo, deberá ser elegido (previa investigación y evaluación) de entre una de las siguientes dos alternativas:

- Mapping File Into Memory
- Unlocked Stream Operation

Si bien solo debe ser elegida e implementada una opción, el alumnado debe ser consciente de ambas ya que podrán ser preguntadas en el coloquio como así también el criterio por el cual se ha elegido la alternativa implementada. Junto con la opción que elija, el grupo deberá investigar y utilizar una de la siguientes funciones, según la opción elegida previamente:

- *posix_madvise* para Mapping File Into Memory
- *posix_fadvise* para Unlocked Stream Operation

La utilidad de la misma, y la forma en la que fue empleada, será evaluada durante el coloquio.

Finalmente el grupo deberá evaluar, seleccionar e implementar dos algoritmos de planificación de disco (uno que genere inanición, y otro que la evite) de entre cualquiera de los siguientes: N-STEP-SCAN, F-SCAN, C-SCAN, SSTF, SCAN, LOOK, C-LOOK.

Consola

El Proceso Disco cuenta con un proceso consola, el cual será iniciado mediante la combinación de funciones fork() y exec(). Una vez iniciada la consola, esta se conectará mediante Sockets Unix al Proceso Planificador del Disco. Su funcionalidad es la de proveer una herramienta para interactuar directamente con el Planificador de Disco. Esta consola podrá ser accedida por teclado una vez conectada al Proceso Planificador de disco y deberá soportar al menos los siguientes comandos:

- *info*: Imprimirá en pantalla la posición actual del cabezal

- *clean <sector inicial> <sector final>*: Limpia los sectores seteando todos sus bytes a '\0', desde el sector inicial hasta el sector final, ambos inclusive.

- *trace [lista sectores]*: Dado una lista de sectores lógicos (entre 1 y máximo 5), por cada uno de estos nos debe imprimir la información de planificación resultante de procesar dicho sector en formato: *<n° de pista>: <n° de sector dentro de la pista>*. Ej:

  :~$ trace 10 11

Posición actual: 0:0
Sector Solicitado: 0:10
Sectores Recorridos: 0:1, 0:2, 0:3, 0:4, 0:5, 0:6, 0:7, 0:8, 0:9, 0:10
Tiempo consumido: 20ms
Próximo Sector: 0:11

Posición actual: 0:10
Sector Solicitado: 0:11
Sectores Recorridos: 0:11
Tiempo consumido: 2ms
Próximo Sector: ...

En caso de que el disco no esté haciendo nada, la información del trace será sólo la de los sectores solicitados. Si se ejecutara un trace mientras se esté atendiendo pedidos del Proceso File System podrán verse resultados distintos como que la "Posición actual" o "Próximo Sector" no sean los que se indicaron en el comando sino que sean de los sectores con los que se encuentra operando el Proceso FileSystem.

La cantidad de información transmitida por este comando puede resultar grande, por lo que el grupo tendrá que encontrar la forma de optimizarlo para transferir la menor cantidad de información posible.

Es importante recalcar que si bien el proceso simula un disco real dados sus parámetros físicos, esta información será únicamente mostrada en pantalla. De ninguna manera se pretende que el tiempo de respuesta de atender un pedido coincida con los tiempos calculados, ***con excepción de los tiempos de lectura y/o escritura de un sector****, los cuales están especificados en el archivo de configuración y* ***sí tienen que ser respetados****..*

Datos de Configuración

- Modo de Inicio [CONNECT / LISTEN]

  En modo CONNECT, el disco realiza la función de acoplarse al Proceso RAID
  - Dirección IP y puerto TCP del Proceso RAID

  En modo LISTEN, el disco espera a que el Proceso File System se conecte
  - Dirección IP y puerto TCP de escucha de conexiones

- Algoritmo de planificación

- Path para el socket Unix.

- Flag para activar o desactivar los logs.

- Identificador del Disco

- CHS

- Tiempo de Lectura en ms
- Tiempo de Escritura en ms
- RPM
- Tiempo de salto de pistas en ms

6.- Ciclo de Desarrollo

Con Ciclo de Desarrollo nos referimos a una planificación estipulada por la Cátedra, mediante la cual se establecen los plazos y objetivos ideales para los cuales los grupos deberían desarrollar el Trabajo Practico. Ésto quiere decir, que se establecen una serie de puntos de control -***checkpoints***- los cuales tienen un conjunto de objetivos a desarrollar y un plazo en el que deberían estar finalizados. No se debe confundir un checkpoint con "entrega", ya que esta ultima implica de manera obligatoria tener todos los objetivos cumplidos para la fecha fijada.

Los checkpoints, son un punto de referencia para el alumnado y para que este comprendan las dificultades y el tiempo que demandarían cada uno de los objetivos. A su vez, el puntual cumplimiento de los checkpoints le permitiría al grupo un desarrollo mas distribuido y progresivo.

Los checkpoints serán seguidos por los ayudantes asignados a cada grupo, a excepción de los que son presenciales. Esto se encuentra explicado en mayor detalle en las **Normas de Trabajo Practico**.

El desarrollo de cada uno de los tres procesos esta planificado de manera paralela, por lo que los tres comenzarán su desarrollo en el primer checkpoint y finalizarán antes de la entrega final o coincidiendo con esta. Esto permite distribuir el desarrollo de manera mas uniforme y balanceada entre todos los integrantes del grupo. Cada uno de los tres procesos tiene una carga, la cual implica cuantos desarrolladores debería, aproximadamente, tener trabajando sobre este a lo largo de todos los checkpoints:

Proceso File System: 2.5
Proceso RAID: 1
Proceso Planificador de disco: 1.5

Este peso no indica que las asignaciones sean fijas, sino que puede haber rotaciones dentro del grupo, lo cual seria una situación ideal.

Por ultimo cabe aclarar que pueden existir tareas fuera de lo ponderado anteriormente y que queda a discreción del grupo su manejo y planificación. Como puede ser el caso de bibliotecas genéricas como: lectura de archivos de configuración o wrappers de sockets que afectan a los tres procesos.

6.1.- Primer Checkpoint

En esta primera etapa comienza el desarrollo de los tres procesos, se considera que al menos una semana y media va a ser utilizada por el grupo para investigar todos los temas relevantes al sistema, definir tareas, definir estructuras y tener un diseño inicial del sistema. El resto del tiempo será utilizo en implementar las estructuras básicas y primeras funcionalidades.

Objetivos:

- **Investigar**
  - FAT32.
  - FUSE.
  - RAID 1.
  - Algoritmos de Planificación de Disco.
  - Métodos de lectura de un archivo.

- **Proceso File System**
  - Leer el Boot Sector.
  - Leer el FAT Region.
  - Obtener clusters libres.
  - Agregar o remover uno o varios clusters libre a una cadena de clusters.
  - Obtener el contenido de un cluster en base al número del cluster.
  - Leer entradas de archivo, directorio o nombre largo a partir de un cluster y todos sus clusters encadenados.
  - Leer el contenido de un directorio y de todos sus subdirectorios.

- **Proceso RAID 1**
  - Implementación de colas de peticiones.
  - Definición de interfaz con Proceso Disco y Proceso File System
  - Algoritmo de distribución entre colas.

- **Proceso planificador de disco**
  - Abrir el archivo disco.
  - Leer un Sector.
  - Escribir un sector.

- **Tareas Generales**
  - Leer archivos de configuración.

Nota

Quien esté desarrollando el Proceso Planificador de Disco puede encapsular las funciones de acceso por sectores al archivo disco dentro de una biblioteca. Esta misma puede ser utilizada inicialmente por el Proceso File System para ir realizando pruebas o ver reflejados los avances sobre este proceso al acceder al archivo disco en vez utilizar una conexión por socket a un proceso que aún no este terminado.

Cuando llegue el momento de implementar sockets en el Proceso File System se puede

conservar el mismo archivo encabezado (.h), haciendo ligeros cambios, de la biblioteca anteriormente utilizada pero cambiar la implementación (el .c) para que en vez de operar directamente sobre el archivo haga las peticiones a través de un socket al Proceso RAID.

Tiempo Estimado	4 semanas
Fecha de Finalización	1/10/2011

6.2.-Segundo Checkpoint [Presencial]

Objetivos:

- **Investigar**
  - Sockets Unix.
  - Sockets TCP.
  - Threads.
  - Semáforos.

- **Proceso File System**
  - Listar Directorios.
  - Leer Archivos.
  - Truncar un archivo.
  - Renombrar un archivo.
  - Consola.

- **Proceso RAID 1**
  - Consola.
  - Conexión con múltiples discos.
  - Asignación de Threads a cada conexión de disco.
  - Distribución de peticiones entre los discos.

- **Proceso Disco** *(Por ahora funciona Sincrónicamente)*
  - Algoritmo de planificación con inanición.
  - Conexión con la Consola.
  - Implementar Consola.
  - Conexión con el Proceso RAID1.

- **Tareas Generales**
  - Implementar el protocolo NIPC.
  - Agregar logueo de información.

Tiempo Estimado	3 semanas

Fecha de Finalización	22/10/2011

6.3.-Tercer Checkpoint

Objetivos:

- **Proceso File System**
  - Escribir Archivos.
  - Modificar Archivos.
  - Conexión con RAID o Disco.
- **Proceso RAID 1**
  - Sincronización de Discos.
- **Proceso Disco**
  - Algoritmo de planificación sin inanición.
  - Disco *Asincrónico*.
- **Tareas Generales**
  - Agregar logueo de información.

Nota: *Para distribuir la carga de una forma mas equitativa, la interfaz mediante la cual el Proceso File System accede al Proceso RAID o el Proceso Disco puede ser encapsulada dentro de una biblioteca y desarrollada por alguien distinto a quien este desarrollando el Proceso File System. Aun así esto queda a criterio del grupo.*

Tiempo Estimado	2 semanas
Fecha de Finalización	5/11/2011

6.4.- Entrega Final

Gran parte de este checkpoint involucra el stress-testing del sistema distribuido completo, como así también la optimización del mismo y los últimos ajustes.

Objetivos:

- **Investigar**
  - Signals.
- **Proceso File System**
  - Cache.
  - Sincronización de múltiples accesos al File System.
  - Crear Archivos.
  - Crear Carpetas.

- **Tareas Generales**
    - Integración de todas las partes.
    - Agregar logueo de información.

Tiempo Estimado	2 semanas
Fecha de Finalización	19/11/2011

7.- Requerimientos técnicos y limitaciones

Compilación

- Se deberá utilizar el compilador gcc bajo el estándar C99.

Comunicaciones – Sockets

- Para la comunicación cliente/servidor se utilizarán los protocolos especificados en el trabajo práctico y se implementarán utilizando sockets orientados a la conexión del tipo AF_INET para IPv4.
- Para el caso particular del Proceso Disco con la consola la conexión debe ser del tipo AF_UNIX.

Se deberá investigar las diferencias técnicas entre Sockets Unix y Sockets tradicionales, ya que ésto mismo podrá ser evaluado durante el coloquio.

Cuando sea necesario utilizar una función para multiplexar I/O, el grupo deberá optar por utilizar alguna/s de las siguientes opciones:

- select
- poll
- epoll

Siendo requerido, para cualquiera de los casos, el conocer las diferencias técnicas o implementaciones de cada una de ellas.

Tipos de Dato y Arquitecturas

Para tratar de asegurar una mayor portabilidad entre arquitecturas, el alumnado deberá evitar el uso del tipo **int**, donde sea necesario tener un tamaño fijo de datos, ya que este mismo cambia su tamaño dependiendo de la arquitectura. En reemplazo se deberá investigar un tipo de dato equivalente pero que asegure un tamaño sin importar la arquitectura que se maneje.

Process Managment y Multithreading

En caso de ser requerido, solo se podrá utilizar la API de la biblioteca pthreads que forma parte del estándar POSIX (NPTL – Native POSIX Thread Library).

Restricciones

Queda prohibido el uso de cualquier mecanismo de exception handling (__try __except) o funciones/bibliotecas de C++.

8.- Anexos

Anexo A :: Archivo Log y Debugging

Manejo de Errores

La aplicación mostrará por consola y generará en el correspondiente archivo Log el código de error y la descripción de dicho código obtenido del sistema para la ejecución fallida de cualquier función del sistema invocada. Para los errores pertinentes a la aplicación se deberá respetar las normas de logueo del trabajo práctico.

Formato del archivo Log

Todos los archivos de Log deberán respetar un mismo formato de presentación. En caso de utilizar distintos niveles detalle, el cambio entre uno u otro debe ser configurable por el usuario.

Se le recomienda al alumno registrar en este archivo **los eventos más importantes de la ejecución de la aplicación**, así como **los valores necesarios para conocer el estado del sistema** en un determinado momento. Esto es muy importante ya que en instancias finales de evaluación es probable que se haga uso de este archivo en situaciones donde la aplicación falla o se requiera hacer un seguimiento del la ejecucion, por lo que este tiene que ser legible.
El formato a respetar es el siguiente:

> *[FECHA] [NOMBREPROCESO] [PIDPROCESO][THREAD ID]: TIPO DE LOG: DATO*

Descripción

- ***Fecha***
  Fecha del sistema. Deberá respetar el siguiente formato [HH:mm:ss.SSS].

- ***Nombre Proceso***
  Nombre del proceso que está escribiendo en el Log.

- ***PID Proceso***
  Process ID del proceso que está escribiendo en el Log.

- ***Thread ID***
  ID del thread que escribe en el archivo. Opcional para el thread principal del proceso.

- ***Tipo de Log***
  INFO, WARN, ERROR ó DEBUG nivel de detalle según lo que consideren apropiado.

- ***Data***
  Descripción del evento ó cualquier información que se considere apropiada.

Eventos de logueo obligatorio

Algunos eventos deberán ser logueados obligatoriamente con cierta información de contexto, a saber:

Proceso File System

- Lista de clusters de un archivo cuando este es abierto.

Proceso RAID

- Conexión y desconexión de nuevos discos.
- Cambios de estados en el RAID.
- Llegada de una petición y Disco donde es procesado.

Proceso Planificador de Disco

Este proceso logueara toda su información unicamente en un archivo. No deberá imprimir nada por consola, solo la consola logueara por stdout.

- Llegada de un pedido, junto con la información de este.
  - Tipo (lectura / escritura)
  - Sectores

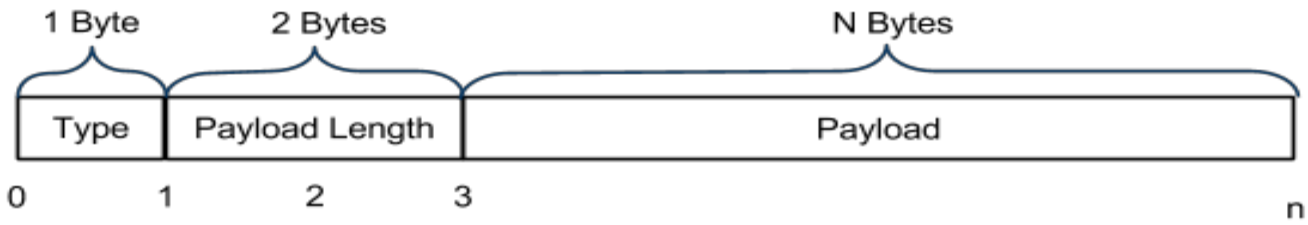
- Información de planificación cuando una petición es procesada.

  Cola de Pedidos: [10, 11, 12 … 50, 51, 52] Tamaño: 30
  Posición actual: 0
  Sector Solicitado: 10
  Sectores Recorridos: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
  Tiempo consumido: 16ms
  Próximo Sector: 11

Anexo B :: Protocolos de comunicación

Protocolo NIPC – (Network Inter-Process Comunication)

Este protocolo se utilizará para comunicar todos los procesos a través de paquetes. Un paquete de datos es una **unidad fundamental de transporte de información** en todas las redes de computadoras modernas. Los paquetes al ser una unidad fundamental, no pueden ser divisibles. Por lo que no se puede descomponer un paquete en varias partes y enviarlas por separado. Nuestro protocolo se maneja a nivel bytes, los cuales tienen el siguiente significado:



- ***Type***
  Identificador que determina el tipo de paquete que es. Esto hace referencia a la información que está contenida en el Payload.

- ***PayLoad Length***
  La longitud en bytes del campo siguiente, es decir del Payload.

- ***Payload***
  Este campo de bytes de longitud indefinida, contiene la información que se necesita enviar.

Extensión del Protocolo NIPC

El protocolo especificado anteriormente, especifica como deberá estar compuesto minimamente cada mensaje, pero esto no indica que no puede ser ampliado. El grupo puede tomar decisiones sobre agregar nuevos campos al protocolo para facilitar su desarrollo. Esto cambios deben ser especificados en un documento y presentado al ayudante asignado durante el desarrollo y a los evaluadores cuando llegue el momento del coloquio.

Handshake

Handshaking ("apretón de manos") es un proceso de negociación que establece de forma dinámica los parámetros de un canal de comunicaciones establecido entre dos entidades antes de que comience la comunicación normal por el canal. Por lo general, un proceso que tiene lugar cuando un equipo está a punto de comunicarse con un dispositivo exterior a

establecer normas para la comunicación. Esto quiere decir que una vez establecida la conexión y antes de empezar con la comunicación normal, tanto el cliente como el servidor se identifican y negocian parámetros de comunicación.

Para el trabajo práctico la funcionalidad del Handshake será la de negociar la comunicación entre dos procesos, haciendo que estos se identifiquen y se informen los parámetros iniciales.

Proceso Disco -> Proceso RAID

El Proceso Disco enviara un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload: El identificador del disco

El Proceso RAID responder con un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload:
    - Vacío, por lo que Payload Length sera 0, en caso de que la conexión se correcta
    - Código/Mensaje de error indicando que la negociación fue fallida y el motivo de esto. Luego de enviar este paquete el Proceso RAID cierra la conexión.

Proceso File System -> Proceso RAID

El Proceso File System enviara un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload: Vacio

El Proceso RAID responder con un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload:
    - Vacío, por lo que Payload Length sera 0, en caso de que la conexión se correcta
    - Código/Mensaje de error indicando que la negociación fue fallida y el motivo de esto (Como puede ser que el Proceso RAID1 aun no este activo). Luego de enviar este paquete el Proceso RAID1 cierra la conexión.

Proceso File System -> Proceso Disco

El Proceso File System enviara un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload: Vacio

El Proceso Disco responder con un paquete NIPC con los siguientes valores específicos:

- Type: 0 (El 0 indica que el paquete es del tipo Handshake)
- Payload:

- Vacío, por lo que Payload Length sera 0, en caso de que la conexión se correcta
- Código/Mensaje de error indicando que la negociación fue fallida y el motivo de esto. Luego de enviar este paquete el proceso Planificador de Disco cierra la conexión.

Anexo C :: Introducción FAT32

El objetivo de este Anexo es el de explicar, de manera superficial, algunos de los detalles de FAT. Este no es un documento definitivo y completo sino que es una guia de referencia para aclarar algunos detalles de manera mas simple. El Alumnado deberá investigar por su cuenta, en los links indicados, los detalles mas completos y puntuales sobre FAT.

Diseño

Contenido	Sector de Boot	Sector de Informacion del File System	Más sectores reservados (opcional)	File Allocation Table #1	File Allocation Table #2	Region de Datos (Para archivos y directorios) ... (Hasta el final de la particion o el disco)
Tam. en Sectores	(numero de sectores resevados)			(num de FATs)*(sectores por FAT)		NumeroDeClusters*SectoresPorCluster

El sistema de archivos FAT esta compuesto por tres secciones básicas:

1. **Región Resevados:** Esta se encuentra localizada al principio de la partición o disco. El primer sector de esta región reservado , es decir el 0, es el Sector de Boot. Este sector es llamado BIOS Parameter Block, el cual incluye información del File System como así también punteros a la ubicación de otros sectores. El tamaño de esta región esta indicada dentro del Sector de Boot.
   En el sector 1 se encuentra el Sector de Información del File System y en el sector 6 se encuentra el Backup del Sector de Boot.

2. **Región FAT:** Típicamente contiene (aunque puede variar) dos copias de la tabla FAT. En esta se encuentra la lista de clusters que son usados por los archivos y directorios.

3. **Región de Datos:** Este es el lugar donde los archivos y directorios son propiamente almacenados. El tamaño de estos puede ser incrementado arbitrariamente simplemente modificando los punteros de la tabla FAT para agregar mas clusters a la lista de sectores asociados al archivo o directorios. Tipicamente la tabla del Directorio Raíz se encuentra en el clusters 2 (El primer cluster de la Region de Datos)

Entradas Especiales

Existen 2 entradas por defecto que se encuentran dentro de cada directorio nuevo. Estas entrada son las de ‘.’ y ‘..’ Estas entradas, son como cualquier otra entrada de directorio con la diferencia de que no tienen entradas Long File Name (explicadas más adelante) ya que el nombre se encuentra dentro de la misma entrada de directorio.
Una de las particularidades de las mismas es hacia donde apunta el “first cluster”. Para el caso de ‘.’ el first cluster apunta al mismo cluster del contenido del directorio donde esta contenida la entrada ‘.’ y para el caso de ‘..’ el first cluster apunta al cluster directorio del contenido del directorio padre al directorio donde nos encontramos posicionados.

Long File Name Entries

Antiguamente los sistemas de archivos FAT solo permitían nombres de archivos de hasta 8 caracteres y 3 caracteres de extensión. Si bien los sistemas FAT mantuvieron retrocompatibilidad con dicha convención, desarrollaron un mecanismo para poder extender la misma.

Las *Long File Name Entries* son entradas de 32 bytes similares a las tradicionales entradas de directorios y de archivos, incluso el campo de atributos se mantiene en el mismo lugar pero con la diferencia de que la mayoría de los bytes son utilizados para almacenar el nombre del archivo. De los 32 bytes, 26 son utilizados para contener caracteres pero estos se encuentran en formato utf16, que a diferencia del tradicional utf8 los caracteres siempre ocupan 2 bytes. Por lo que en los 26 bytes solo entran 13 caracteres.

El máximo tamaño del nombre de un archivo en FAT32 es de 256 caracteres, para lograr esto simplemente lo que se hace es crearse **n** entradas sucesivas de *Long File Name Entries* las cuales satisfagan el tamaño de nombre deseado.
Las *Long File Name Entries* siempre se encuentran delante de la entrada de archivo o directorio a la que pertenezca el nombre. Con excepción de la entrada "." y ".." todas las entradas siempre poseen al menos una *Long File Name Entry*.

La forma en la que uno puede detectar que se trata de una entrada Long File Name en vez de una entrada de directorio o de archivo es a través del campo de atributos.

Consideraciones

- Para nuestro caso no va a ser necesario gestionar el owner (dueño) de nuestro archivo. Esto quiere decir que cualquier usuario puede acceder a nuestro archivo.

- El nombre de los archivos/directorios incluyendo la extensión no superará los 13 caracteres.

- Los nombres de archivos y/o directorios no contendrán espacios.

- El nombre del archivo está formado por un nombre y una extensión separados por el caracter '.' (punto). El nombre es una cadena de al menos un caracter de tipo alfanumérico (incluidos también el guión medio y el guión bajo). La extensión es una cadena de tres caracteres alfanuméricos**.**

- Si bien el tamaño del disco puede ser de varios GB el File System por simplicidad para el alumno, no se contempla que contenga más de 200 archivos y/o carpetas.

- El campo "Number of free clusters on the drive" del FS Information Sector deberá ser seteado en -1 y no será manipulado. Lo mismo sucede para el campo "Number of the most recently allocated cluster" el cual debera estar en 0xFFFFFFFF.

- Solo será necesario manejar los siguientes tipos de archivo: Archive, Directory/Subdirectory y Long File Name para el campo File Attribute.

Anexo D :: Documentación

The Single UNIX® Specification, Issue 7 from The Open Group
http://pubs.opengroup.org/onlinepubs/9699919799/idx/headers.html

FAT
http://msdn.microsoft.com/en-us/windows/hardware/gg463080
http://en.wikipedia.org/wiki/File_Allocation_Table

FUSE
http://fuse.sourceforge.net/

Se recomienda el siguiente material bibliográfico como soporte técnico:

- The C Programming Language de Kernighan & Ritchie
- Linux Programming Unleashed de Kurt Wall
- Linux System Programming: Talking Directly to the Kernel and C Library de Robert Love

Foro de Consulta:
http://www.campusvirtual.frba.utn.edu.ar/especialidad/course/category.php?id=54

Wiki de la Cátedra
http://www.tpsosutnfrba.com.ar/wiki