Ingeniería en Informática

Sistemas Operativos

Trabajo Investigación

Modo de manejo Ext2

Equipo Docente	Nicanor Casas
	Graciela De Luca
	Waldo Valiente
	Gerardo Puyo
	Sergio Martín
	Federico Díaz
	Sebastián Deuteris

GRUPO: 1
Días de Cursada: *Lunes y Jueves*

Alumnos		
Apellido	**Nombre**	**DNI**
Narmona	Alejandro	30862765
Maisano	Hernando	27728518
Cescon	Javier	31632108

Acreditación:

Instancia	Fecha	Calificación
PRE-ENTREGA	27/06/2012	
FINAL	11/07/2012	

Contenido

1. Objetivo

Este trabajo práctico tiene como objetivo conocer el sistema de archivos EXT2 (Second Extended File System), a fin de poder implementar dicho sistema de archivos en el Sistema Operativo SODIUM.

Veremos además, como es la interrelación y el flujo de comunicación entre el nuevo File System a implementar y SODIUM, a través de la interfaz denominada Virtual File System (VFS).

Un sistema de gestión de archivos es aquel sistema de software que proporciona a los usuarios unos servicios relativos al empleo de archivo, esto acaba con la necesidad para el usuario o programador de desarrollar software de propósito específico para cada aplicación.

Dentro del alcance se encuentra el poder acceder a una partición Ext2, poder conocer el contenido de la misma, poder crear carpetas, crear archivos, poder manipularlos, lograr desmontar la unidad y finalmente poder acceder a la misma desde otro sistema operativo que reconozca Ext2.

2. Resumen

A continuación se describe un breve resumen del funcionamiento del sistema de archivos implementado en SODIUM.

El sistema de archivos fue diseñado por Remy Card en 1993 para Linux fue denominado EXT, actualmente el mismo va por la versión 4, pero mantiene compatibilidad con EXT2, que fue el que se implementó en SODIUM

El sistema de archivos EXT permite a SODIUM o cualquier sistema operativo que lo implemente, mantener los archivos en una estructura de un único árbol del sistema en la que los nodos del árbol son directorios y los nodos hoja son los archivos. La información de los nodos del árbol se mantiene en las estructuras inodo, y la estructura del sistema de ficheros en las estructuras superbloque.

El sistema de archivo EXT2 se construye con la idea de que los datos contenidos en los ficheros se guarden en Bloques de Datos. Estos bloques de datos son todos de la misma longitud y, si bien esa longitud puede variar entre diferentes sistemas de ficheros EXT2 el tamaño de los bloques de un sistema de archivos EXT2 en particular, se decide cuando se crea.

Cuando un proceso de usuario hace una llamada al sistema de archivos, esta es atendida por el Virtual File System (VFS), y posteriormente este dirige la llamada al sistema de archivos particular de que se trate. Al momento de comenzar a trabajar con este nuevo sistema de archivos, ya se encontraban implementados FAT12, FAT16 y FAT32 con el VFS.

Cuando SODIUM inicia, se monta el sistema de archivos seleccionado al momento de ejecutar el script configurar.sh. En nuestro caso, montaremos la imagen “imgEXT” generada desde Linux y formateada con el sistema de archivos EXT2.

Luego, al momento de levantar dicha imagen (se carga toda la imagen en memoria RAM mediante el ramfs), se cargarán las estructuras necesarias para el funcionamiento del FS (File System) EXT2.

Las estructuras necesarias a lo largo de toda la utilización del FS son:

- Superbloque: Contiene información clave del sistema, tanto estática (tamaño de las estructuras) como dinámica (espacio libre). Tan importante es esta estructura que se mantienen copias de ella desperdigadas por el disco.

Nombre	Descripción
s_inodes_count	Cantidad de Inodos
s_blocks_count	Cantidad de Bloques
s_r_blocks_count	Cantidad reservada de bloques
s_free_blocks_count	Cantidad de bloques libres
s_free_inodes_count	Cantidad de inodos libres
s_first_data_block	Primer bloque de datos
s_log_block_size	Tamaño de bloque
s_log_frag_size	Tamaño de fragmento
s_blocks_per_group	Cantidad de bloques por grupo

s_frags_per_group	Cantidad de fragmentos por grupo
s_inodes_per_group	Cantidad de inodos por grupo
s_mtime	Fecha del último montaje
s_wtime	Fecha de la última escritura
s_mnt_count	Cantidad de montajes desde la última comprobación
s_max_mnt_count	Cantidad máxima de montajes entre comprobaciones
s_magic	Magic
s_state	Estado del sistema de archivo
s_errors	Comportamiento cuando ocurre un error
s_minor_rev_level	Número menor de revisión
s_lastcheck	Fecha de la última comprobación
s_checkinterval	Máximo intervalo de tiempo entre comprobaciones
s_creator_os	Sistema Operativo
s_rev_level	Número de revisión
s_def_resuid	uid del superusuario por defecto
s_def_resgid	gid del superusuario por defecto
s_first_ino	Primer inodo no reservado
s_inode_size	Tamaño de la estructura de inodo
s_block_group_nr	Número de grupo que contiene éste superbloque
s_feature_compat	Indicador de características compatibles
s_feature_incompat	Indicador de característica incompatibles
s_feature_ro_compat	Compatibilidad con sólo lectura
s_uuid[16]	uuid 128-bit para el volumen
s_volume_name[16]	Nombre del volumen
s_last_mounted[64]	Directorio sobre el que se montó por última vez
s_algorithm_usage_bitmap	para compresión
s_prealloc_blocks	Número de bloques que debe intentar preasignarse
s_prealloc_dir_blocks	Número de bloques a preasignar para directorios
s_padding1	Número de bloques de relleno
s_journal_uuid[16]	uuid del superblock journal
s_journal_inum	Número de inodo del archivo journal
s_journal_dev	Número del dispositivo journal
s_last_orphan	Inicio de la lista de inodos a borrar
s_hash_seed[4]	Semilla hash de HTREE
s_def_hash_version	Versión por defecto de hash a usar
s_reserved_char_pad	Carácter de relleno por defecto
s_reserved_word_pad	Palabra de relleno por defecto
s_default_mount_opts	Opciones de montaje por defecto
s_first_meta_bg	Primer grupo de bloques
s_reserved[190]	Reserva de bloques

- Descriptores de grupo: Estructura que describe los contenidos de un grupo de bloques.

Nombre	Descripción
bg_block_bitmap	Dirección del Bloque del bitmap de bloques
bg_inode_bitmap	Dirección del Bloque del bitmap de inodos

bg_inode_table	Dirección del Bloque de la tabla de inodos
bg_free_blocks_count	Número de bloques libres
bg_free_inodes_count	Número de inodos libres
bg_used_dirs_count	Número de directorios
bg_pad	No utilizado
bg_reserved[3]	Reservado para futura extensión

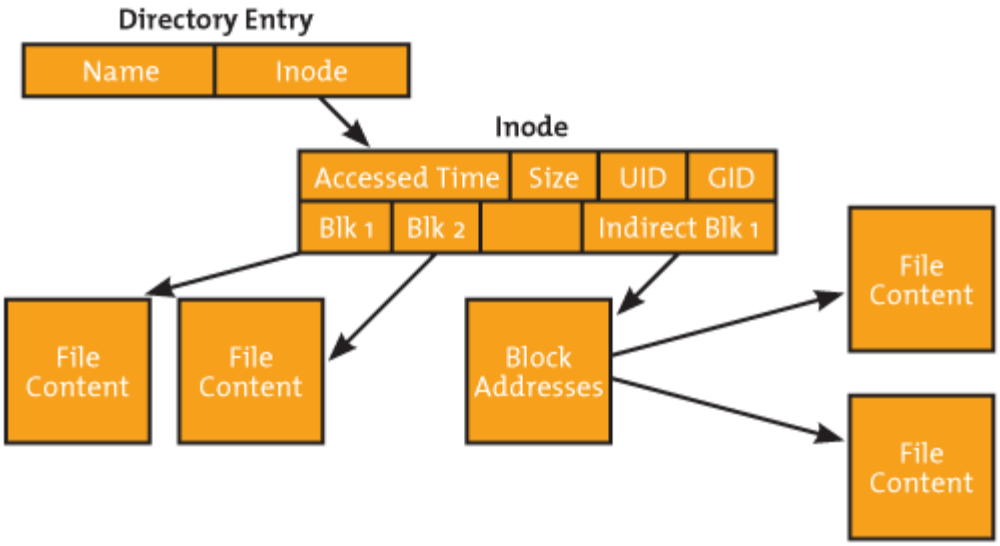
- Entrada de Directorio: Contiene el nombre del archivo o directorio, el número de inodo al que apunta (el cual tendrá toda la información necesaria para su utilización) y el tamaño del mismo.

Nombre	Descripción
inode	Número de inodo
rec_len	Longitud de Directory
name_len	Largo del nombre
file_type	Tipo de archivo
name	Nombre de archivo

- Inodo: El i-nodo contiene la información de cada archivo: tipo de archivo, accesos, propietario, tamaño, puntero a bloques de datos, etc. Las direcciones de los bloques de datos asignados a un archivo se almacenan en su i-nodo.

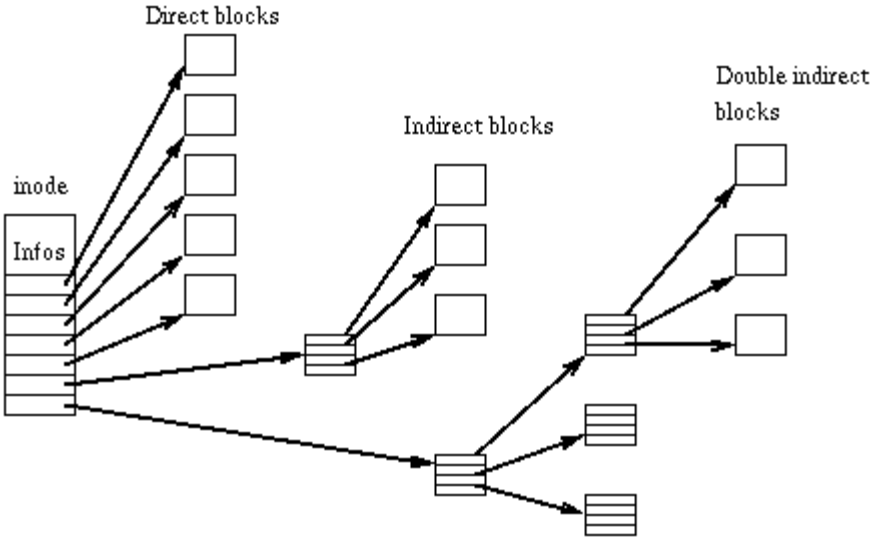
Nombre	Descripción
i_mode	Formato del fichero y permisos de acceso
i_uid	Identificador del propietario
i_size	Tamaño del archivo en bytes
i_atime	Fecha del último acceso al archivo
i_ctime	Fecha de creación del inodo
i_mtime	Fecha de última modificación del archivo
i_dtime	Fecha de supresión del archivo
i_gid	Identificador de grupo
i_links_count	Número de enlaces que apuntan al inodo
i_blocks	Número de bloques asignados para contener los datos del inodo
i_flags	Comportamiento al acceder al inodo
i_block	Direcciones de bloque de datos asociados al inodo
i_generation	Número de versión asociado al inodo
i_file_acl	Dirección del descriptor de la lista de control de acceso asociada al archivo
i_dir_acl	Dirección del descriptor de la lista de control de acceso asociada a un directorio
i_faddr	Dirección del fragmento del archivo

En el siguiente gráfico se muestra la relación entre las entradas de directorio (Directory Entry) y los Inodos:



Design and Implementation of the Second Extended File system (http://web.mit.edu/tytso/www/linux/ext2intro.html)

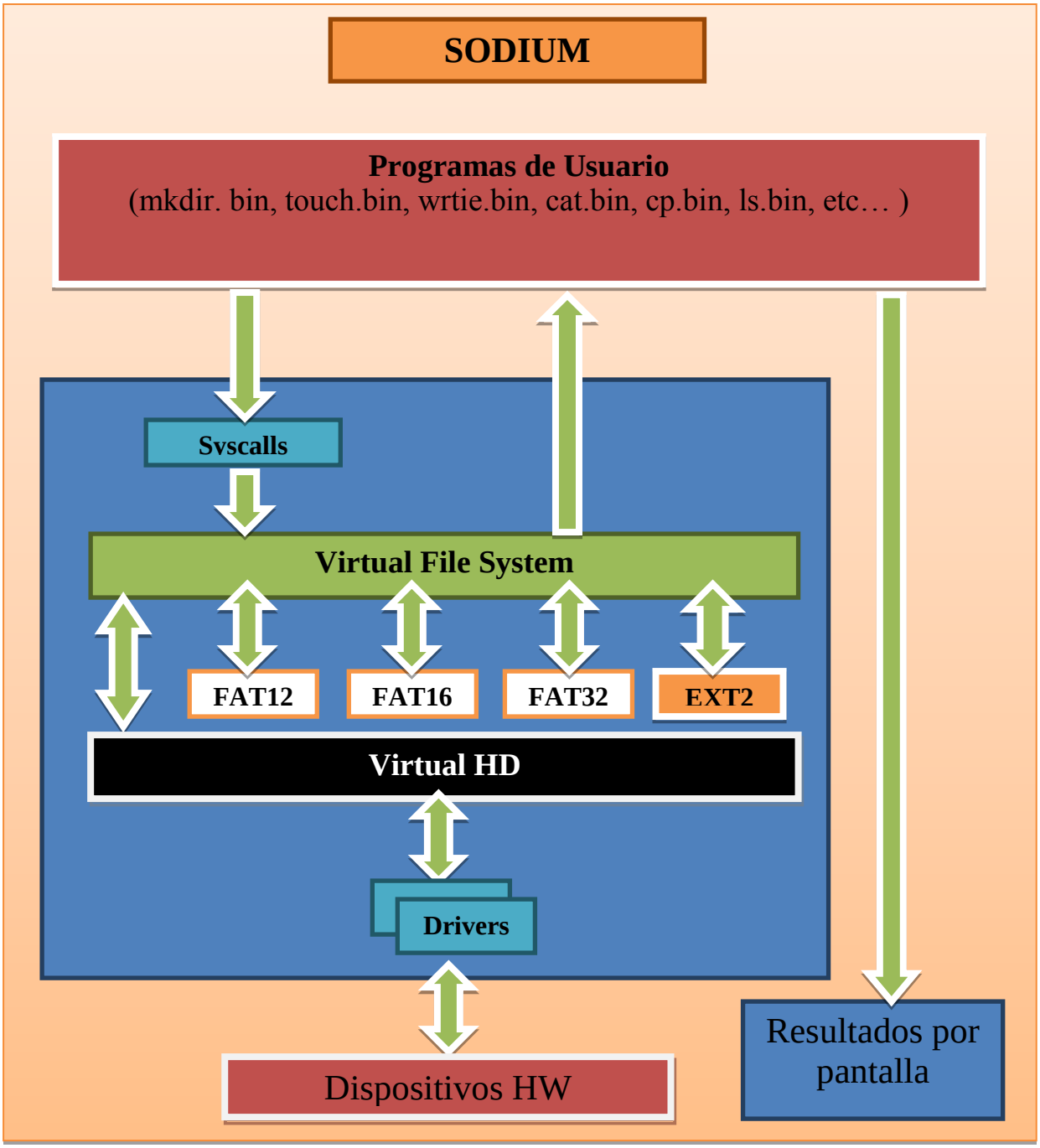
Estructura de almacenamiento de un inodo:



Design and Implementation of the Second Extended File system (http://web.mit.edu/tytso/www/linux/ext2intro.html)

Cuando una aplicación usuario necesita acceder a un archivo, tanto para la lectura como para la escritura, lo hace mediante Syscalls, estas llaman a funciones de VFS y dependiendo el sistema de archivo que está corriendo, el VFS llama a funciones propias de cada sistema de archivos.

A continuación se muestra un gráfico de cómo funciona dicho mecanismo:


SODIUM
Programas de Usuario
(mkdir. bin, touch.bin, wrtie.bin, cat.bin, cp.bin, ls.bin, etc…)
Svscalls
Virtual File System
FAT12
FAT16
FAT32
EXT2
Virtual HD
Drivers
Dispositivos HW
Resultados por pantalla

NOTA: El modulo de Virtual HD (Hard Disk) todavía no fue implementado.

Estructura física

La estructura está formada por un superbloque, que contiene grupos de bloques. Todos los grupos de bloques tienen el mismo tamaño y están colocados de manera secuencial. Cada bloque contiene una copia del superbloque, una copia de la tabla de descriptores de grupo, un bloque de bitmap para los bloques, u bloque de bitmap para los i-nodos, una tabla de i-nodos y bloques de datos.

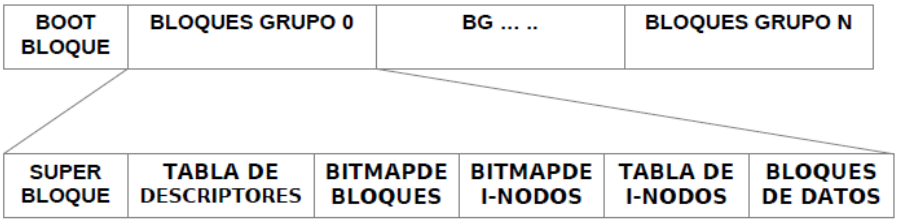


Diagrama de lo Superbloque y Bloques utilizado en Ext2 (Fuente: Presentación de Superbloque - Ext2. Diseño de Sistemas Operativos. Universidad de Las Palmas de Gran Canaria)

Los tamaños de los distintos bloques se determinan por el tamaño de bloque, normalmente de 1024 bytes y puede ser hasta 4096, la información del tamaño del bloque se encuentra el Superbloque

Nombre	Tamaño
Boot Sector	1024 bytes
Super Bloque	1 bloque
Tabla de Descriptores	N bloque
Bitmap de Bloques	1 bloque
Bitmap de I-Nodos	1 bloque
Tabla de I-Nodos	1 bloque
Bloque de Datos	N bloque

En el caso de la Tabla de Descriptores y el Bloque de Datos, la cantidad de bloques es asignada al momento de dar formato a la unidad, dependiendo del tamaño disponible y del tamaño de bloque dicha cantidad se encuentra en el SuperBloque.

2.1. Introducción

El presente documento expone las funciones implementadas dentro de SODIUM para el funcionamiento de un nuevo sistema de archivos, en este caso EXT2. Asimismo, describiremos como se implementó dicho sistema de archivos en SODIUM.

En los siguientes puntos describiremos las posibles mejoras a implementar, asi como también una conclusión del presente trabajo práctico.

2.2. Resumen de modificaciones

2.2.1. Configurador

Al configurar.sh se le agregó a las opciones de File System EXT2. Al seleccionar esta opción se crea una imagen EXT2.

2.2.2. Scripts

Herramientas/listar_binarios_usuario.sh se modifico para que no este hardcodeado el nombre de la imagen del File System, sino que sea una variable. De esta forma para EXT utiliza una imagen y FAT otra.

2.2.3. Makefile

Makefile se le agregó la copia de la imagen Ext2 al path de donde se realiza el build.

kernel/Makefile para poder compilar los nuevos archivos creados para ext2 se tuvo que agregar al mismo que OBJETOS_FS necesita SODIUM.

kernel/Makefile también comentamos la compilación (los archivos .o) de los módulos de fat12, 16 y 32 ya que para las pruebas de EXT2 necesitamos mucha memoria para realizar las pruebas de comandos (ver además VFS).

build/Makefile y solo/Makefile se puso una condición para que, dependiendo el sistema de archivo, seleccione la imagen FAT o la imagen EXT2.

2.2.4. Variables

2.2.4.1. Tipos.h

Se necesito agregar enteros de 64 bits y por ello se creó el tipo de dato u64 (unsigned long).

2.2.4.2. FS_DEF.H

Se agregaron define de errores para EXT2.

2.2.5. Programas

2.2.5.1. Comando LS

Se incluyeron mejoras al programa ls haciendo que estén todas las columnas alineadas y agregándole color azul a los directorios.

2.2.5.2. Comando CAT

Se detecto un error en el mismo, ya que al ejecutarlo traía "más caracteres" que debería traer por el peso y contenido del archivo. Se verifico que al Leer lo hacia de a 10 caracteres siempre, entonces siempre devolvía al VFS múltiplo del 10 caracteres. Se modifico para que lea de a 1 carácter solucionando este inconveniente.

2.2.5.3. Find

Se modificó el comando find para que si no se le pasa la ruta del directorio en donde buscar, se tome la ruta en donde se está parado (ruta actual). Esto no estaba implementado en FAT.

2.2.5.4. VFS

Se hicieron modificaciones sobre el Virtual File System para que funcione correctamente EXT2

- Agregado de file system EXT2, llamando a la función "vFnAgregarFileSystemType".
- Se modificó la función "vFnAgregarFileSystemType", para llamar a la función que carga la estructura de SuperBloque de EXT2
- Se creó la función "vFnExt_CargarSBEXT2" que es la encargada de asignarle las funciones específicas del sistema de archivos a la estructura de operaciones como también el encargado de leer el SuperBloque del File System.
- Adicionalmente se modificó que la función "pcFnLeerPBP" para que determine correctamente el sistema de archivos, ya que antes no detectaba correctamente en caso que sea FAT32.
- Por problemas de escases de memoria, eliminamos la compilación de los módulos de FS de fat12, 16 y 32. Lo que se hizo fue "comentar" las líneas de código de las llamadas a las funciones de operaciones del VFS en cada una de las funciones vFnFat_CargarSBFATXX (XX según el tipo de FAT, 12, 16 y 32). No modificamos la función vFnFat_CargarSBFAT12B que correspondía al antiguo fat12 de SODIUM.

2.2.5.5. RAMFS

Se crearon 2 funciones para poder Leer o Escribir EXT2 en el RAMFS ya que las funciones existentes usaban internamente la variable del tamaño del bloque de FAT. Dichas funciones es genéricas, y no hay constantes dentro, leen o escriben desde una posición hasta un tamaño dado.

- Se creó la función iFnLeerSectorExt, para poder leer el RamFs, o sea el disco de SODIUM actualmente.
- Para mantener el sistema estable se necesito además la función iFnEscribirExt similar a la de Leer.

2.2.5.6. Reloj

Se agregaron 2 funciones para poder asignar los campos necesarios de fecha en el formato Ext2 (número que cuenta la cantidad de segundos desde 01/01/1970)

- Se creó la función vFnObtenerFechaEnLong que recibe una fecha y devuelve el número que representa esa fecha.
- Se creó la función vFnObtenerFechaActualEnLong devuelve el número que representa la fecha y hora actual.

2.3. Resumen de agregados

2.3.1. commonExt

Aquí se encuentran las funciones, variables y constantes que sirven para todos los distintos sistemas de archivos Ext tanto Ext2, Ext3 y Ext4.
Dentro se encuentran las siguientes funciones:

2.3.2. ext2_fs

Aquí se encuentran las funciones, variables y constantes que sirven sólo para Ext2. Aquí se encuentra la definición del inodo, estructura de directorio, estructura de descriptor de grupo, estructura de super bloque y la estructura de Info del File System EXT2.

2.4. Funciones

2.4.1. commonExt

int	**iFnBuscarBarraExt** (char *pcDirectorio, char *pcRuta, int iPosInicio)
	busca en el path si hay barra "/"
void	**vFnObtenerExtensionExt** (char *pcExtensionExt, char *pcNombre)
	Obtiene la extension de un nombre de archivo.
void	**vFnConvertirUnixFecha** (int iFechaEnFormatoLinux, struct **stuFecha** *pstuFechaRetorno)
	Convierte un tipo fecha de unix a estructura de fecha de Sodium.
void	**vFnConvertirUnixHora** (int iFechaEnFormatoLinux, struct **stuHora** *pstuHoraRetorno)
	Convierte un tipo fecha de unix a estructura de hora de Sodium.
char *	**pcFnObtenerPermisosDeArchivo** (int iModo)
	Obtiene en forma de cadena de texto los permisos de un archivo.
int	**iFnLetraTipoArchivo** (int iModo)
	Segun el modo obtiene la letra que representa para imprimir en el LS.
char *	**pcObtenerTipoElementoPorNumero** (int iTipoElemento)
	Obtiene el tipo de Elemento por el Numero.
int	**iFnValidarNombreElemento** (char *pcNombreElemento)
	Valida que el nombre del Elemento contenga caracteres validos.

2.4.2. ext2_fs

int	**iFnMontar_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque)
	Ejecuta el comando Mount para RAMFS de EXT2.
int	**iFnLeerSB_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque)
	Carga la estructura stuBootSector que contiene el sector de booteo de EXT2.
struct **stuInodoExt2** *	**pstuInodoExt2FnObtener** (struct **stuInfoSuperBloqueExt2**

	*pstuInfoSuperBloqueExt2_SuperBloque, **u32** u32NroInodo)
	Obtiene un inodo a travez del numero de inodo Equivalente a la funcion de linux: ext2_get_inode.
struct **stuDescriptorGrupoBloques** *	**pstuDescriptorGrupoBloquesFnObtener** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, unsigned long ulGrupoBloque)
	Obtiene el Descritor de Grupo (DG) a travez del numero de DG.
int	**iFnCd_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcRuta)
	Ejecuta el comando cd, cambia de directorio actual.
int	**iFnBuscarElemento** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcRuta, **u32***pu32NroNodoOrigen, char *pcUltimoNombreElemento, int iTraerUltimoInodo)
	Busca un elemento en un path pasado por parámetro.
int	**iFnBuscarDir_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcRuta, int iTipo, int iMostrarMensajeError)
	Valida que el contenido de un path sea valido.
int	**iFnDesmontar_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque)
	Ejecuta el comando desmontar para un sistema de archivos ext2, Libera la memoria asignada a las estructuras de EXT 2 a desmontar.
int	**iFnListarElementos_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcRuta, struct **stuListaArch** **ppstuListaArch_Archivos)
	Ejecuta el comando ls para ext2.
int	**iFnCrearDirectorio_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombre)
	Crea un directorio nuevo y los directorios ficticios "." y ".." del directorio nuevo.
unsigned char *	**pucObtenerDirectorioVacio** (**u32** u32NroInodoActual, **u32** u32NroInodoPadre)
	Crealos directorios ficticios "." y ".." del directorio nuevo.
int	**iFnObtieneNroDescriptorGrupoActual** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, **u32**u32NroInodoPadre)
	Obtiene el nro del Descriptor de Grupo (DG) para un Inodo Dado dentro del mismo DG.
unsigned long	**ulFnObtieneNroDescriptorGrupoOtro** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, **u32**u32NroInodoPadre)
	Obtiene el nro del Descriptor de Grupo (DG) para un Inodo Dado en cualquier DG.
int	**iFnBuscarBitLibre** (unsigned char ucCaracterASetear)
	Busca un bit en 0 del Bitmap y devuelve la posicion del char Equivalente a la funcion de linux: ext2_find_next_zero_bit.
int	**iFnSetBitOcupado** (**u32** u32NroInodo, unsigned char *pucCaracterASetear)
	Setea en 1 (ocupado) un bit del Bitmap que corresponda Equivalente a la funcion de linux: ext2_set_bit_atomic.
int	**iFnSetBitLibre** (**u32** u32NroInodo, unsigned char *pucCaracterASetear)
	Setea en 0 (libre) un bit del Bitmap que corresponda.
int	**iFnObtenerNuevoInodo** (int **iModo**, struct **stuInodoExt2** *pstuInodoExt2_Padre, struct **stuInodoExt2** *pstuInodoExt2_Nuevo,**u32** *u32NroInodeNuevo,

	struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, **u32** u32NroInodoPadre)
	Crea (aisgna) un nuevo Inodo Libre (ext2_new_inode).
void	**vFnIncrementarLinksInodo** (struct **stuInodoExt2** *pstuInodoExt2_Inodo)
	Incrementa el numero de links que tiene ese inodo Equivale a la funcion de linux: inode_inc_count.
void	**vFnDecrementarLinksInodo** (struct **stuInodoExt2** *pstuInodoExt2_Inodo)
	Decrementa el numero de links que tiene ese inodo (inodo) Equivale a la funcion de linux: inode_dec_count.
struct **stuEntradaDirectorio2Ext2** *	**pstuEntradaDirectorio2Ext2FnObtenerPorTipo** (struct **stuInodoExt2** *pstuInodoExt2_Inodo, char *pcNombreDir, int iTipoElemento)
	Busca un elemento (dir, archivo, link, etc) segun el nombre y el tipo de elemento Equivale a la funcion de linux: ext2_find_entry.
int	**iFnAgregarEntradaDirectorio** (struct **stuInodoExt2** *pstuInodoExt2_Padre, char *pcNombreDir, **u32** u32NroInodoActual, int iTipoElemento)
	Crear el directorio (inodo del tipo directorio), y asocia crea el nombre del mismo como multiplo de 4 caracteres, de forma que sea performante.
int	**iFnEscribirBloque** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, unsigned char *pucNuevoBloque, int iNroBloque)
	Escribe un Bloque de Datos con el contenido pasado por paramentro.
int	**iFnObtenerBloqueVacio** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, **u32** u32NroInodo, unsigned long *pulNroBloque)
	Obtiene un bloque vacio y lo reserva.
int	**iFnGuardarSuperBloque_EXT2** ()
	Guarda el SB del EXT2.
int	**iFnGuardarBloques_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque)
	Guarda los bloques en el diskette.
int	**iFnCrearArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombre)
	Crear un archivo en EXT2.
u32	**u32FnObtenerFlagsInodo** (int **iModo**, **u32** u32Flags)
	setea los i_flags segun el directorio padre
int	**iFnGuardarDescriptorGrupo** (struct **stuDescriptorGrupoBloques** *pstuDescriptorGrupoBloques_DescGB, int iNroDescriptorGrupo)
	guarda un Descriptor de grupo en Disco(RAMFS)
int	**iFnGuardarInodo** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, struct **stuInodoExt2***pstuInodoExt2, **u32** u32NroInodo)
	guarda un Inodo en Disco(RAMFS)
void	**vFnAsignarNombreDirectorioEXT** (char *pcDirNameOrigen, char *pcDirNameDestino)
	Setea el Nombre del Elemento para Ext llena con espacios hasta multiplo de 4 caracteres.

u64	**u64FnObtieneNroBloqueDatos** (struct **stuInodoExt2** *pstuInodoExt2_Inodo, **u64** ulIndice)
	Busca y Obtiene el Nro de bloque de Datos de un Inodo.
u64	**u64ObtenerNroBloqueDatosIndireccion** (**u64** ulNroBloqueIndireccion, **u64** ulIndice, int iNroIndireccion)
	Obtiene el Nro de bloque de la Indireccion que corresponda.
int	**iFnMostrarArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombreArch, void **ppvDatos)
	Ejecuta el comando cat: mostrar info del archivo.
char *	**pcFnObtieneInfoBloqueDatos** (**u64** ulNroBloque, int iTamanio)
	Obtiene puntero a la informacion de un Bloque de Datos.
int	**iFnEliminarArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombreArch)
	ejecuta el comando rm archivo: elimina archivo
int	**iFnDesasignaBloques** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, struct **stuInodoExt2***pstuInodoExt2_Inodo, **u64** u64NroInodo)
	desasigna los bloques de un inodo
int	**iFnDesasignaInodo** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, struct **stuInodoExt2***pstuInodoExt2_Inodo, **u64** u64NroInodo)
	desasigna un inodo
int	**iFnBorrarElementoPorTipo** (struct **stuInodoExt2** *pstuInodoExt2, char *pcDirName, int iTipoElemento)
	Borra un elemento (dir, archivo, link, etc) segun el nombre y el tipo de elemento Equivale a la funcion de linux: ext2_find_entry.
int	**iFnCopiarArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcPathDestino, char *pcNombreArch, void *pvData, int iTamArch)
	Ejecuta el comando CP para copiar arhivo.
int	**iFnGuardarInfoBloqueDatos** (unsigned char *pucBuffer, **u64** u64NroBloque, int iTamanio)
	Obtiene puntero a la informacion de un Bloque de Datos.
int	**iFnEscribirArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombreArch, char *pcDatos)
	Escribe en un archivo existente la informacion pasada por parámetro.
int	**iFnEliminarDirectorio_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombreDir)
	Elimina un directorio que este vacio.
int	**iFnExistenElementosEnDirectorio** (struct **stuInodoExt2** *pstuInodoExt2_Inodo)
	Busca en un directorio e indica si está o no vacio.
int	**iFnBuscarArchivo_EXT2** (struct **stuInfoSuperBloqueExt2** *pstuInfoSuperBloqueExt2_SuperBloque, char *pcNombre, char *pcRuta)
	Busca un archivo en la unidad montada.

2.5. Casos de ejemplo

2.5.1. Pruebas en EXT2

2.5.1.1. Comando "ls": ls [directorio] [-r]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un directorio con path completo	ls A./dir1	Se debe mostrar el contenido del directorio.	
Se ingresa un directorio	ls dir1	Se debe mostrar el contenido del directorio.	
Se ejecuta el comando sin especificar directorio	ls	Se debe mostrar el contenido del directorio actual	
Se ingresa el nombre de un archivo	ls doc1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	ls dir4	Se debe mostrar el mensaje de error correspondiente	
Se ejecuta el comando sin especificar directorio y con el parámetro -r	ls -r	Se debe mostrar el contenido de los directorios de manera recursiva a partir del directorio actual	
Se ingresa un directorio y el parámetro -r	ls dir1 -r	Se debe mostrar el contenido de los directorios de manera recursiva a partir del directorio ingresado	
Se ingresa un directorio inexistente y el parámetro -r	ls dir4 -r	Se debe mostrar el mensaje de error correspondiente	

2.5.1.2. Comando "cd": cd [directorio]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un directorio con path completo	cd A./dir1	Se debe cambiar el directorio actual al ingresado.	
Se ingresa un directorio	cd dir1	Se debe cambiar el directorio actual al ingresado.	
Se ejecuta el comando sin especificar directorio	cd	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un archivo	cd doc1	Se debe mostrar el mensaje de error correspondiente	

2.5.1.3. Comando "touch": touch [Archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa el nombre con path completo	touch A./dir1/pepe	Se debe crear el archivo en el directorio especificado.	
Se ingresa el nombre	touch pepe	Se debe crear el archivo en el directorio actual.	
Se ejecuta el comando sin ningún nombre	touch	Se debe mostrar el mensaje de error	

		correspondiente	
Se ingresa el nombre de un archivo existente	touch doc1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	touch dir4/pepe	Se debe mostrar el mensaje de error correspondiente	

2.5.1.4. Comando "mkdir": mkdir [directorio]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa el nombre con path completo	mkdir A./dir1/pepe	Se debe crear el directorio en la ruta especificada.	
Se ingresa el nombre	mkdir pepe	Se debe crear el directorio en la ruta actual.	
Se ejecuta el comando sin ningún nombre	mkdir	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un directorio existente	mkdir dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	mkdir dir4/dir2	Se debe mostrar el mensaje de error correspondiente	

2.5.1.5. Comando "rm": rm [Archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un archivo con path completo	rm A./doc1	Se debe eliminar el archivo en la ruta especificada.	
Se ingresa solo el nombre del archivo	rm doc1	Se debe eliminar el archivo en la ruta actual.	
Se ingresa el nombre de un directorio	rm dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un archivo inexistente	rm doc4	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	rm dir4/doc1	Se debe mostrar el mensaje de error correspondiente	

2.5.1.6. Comando "rmdir": rmdir [directorio]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un directorio con path completo	rmdir A./dir1	Se debe eliminar el directorio en la ruta especificada.	
Se ingresa solo el nombre del directorio	rmdir dir1	Se debe eliminar el directorio en la ruta actual.	
Se ejecuta el comando sin ingresar el directorio	rmdir	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un archivo	rmdir doc1	Se debe mostrar el mensaje de error correspondiente	

Se ingresa un directorio inexistente	rmdir dir4	Se debe mostrar el mensaje de error correspondiente	
El directorio contiene archivos	rmdir dir1	Se debe mostrar el mensaje de error correspondiente	

2.5.1.7. Comando "find": find [archivo] [directorio] [-r]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un directorio con path completo	find doc1 A./dir1	Se debe mostrar el resultado de la búsqueda en el directorio especificado.	
Se ingresa solo el nombre del archivo	find doc1	Se debe mostrar el resultado de la búsqueda en el directorio actual.	
Se ejecuta el comando sin especificar el archivo	find	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	find doc1 dir4	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre del archivo y el parámetro -r	find doc1 -r	Se debe mostrar el resultado de la búsqueda de forma recursiva a partir del directorio actual.	
Se ingresa el nombre del archivo, del directorio y el parámetro -r	find doc1 dir1 -r	Se debe mostrar el resultado de la búsqueda de forma recursiva a partir del directorio especificado.	

2.5.1.8. Comando "cat": cat [archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un archivo con path completo	cat A./doc1	Se debe mostrar el contenido del archivo en la ruta especificada.	
Se ingresa solo el nombre del archivo	cat doc1	Se debe mostrar el contenido del archivo en la ruta actual.	
Se ejecuta el comando sin ingresar el archivo	cat	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un directorio	cat dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un archivo inexistente	cat doc4	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	cat dir4/doc1	Se debe mostrar el mensaje de error correspondiente	

2.5.1.9. Comando "cp": cp [Archivo] [Archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa ruta origen y	cp A./doc1	Se debe copiar el archivo	

destino con path completo	A./dir1/doc2	origen en la ruta destino	
Se ingresa ruta origen y destino con path completo	cp doc1 doc2	Se debe copiar el archivo origen en la ruta actual	
Se ingresa solo la ruta origen y destino	cp doc1 dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa solo la ruta origen	cp doc1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un directorio en la ruta origen	cp dir1 dir2	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	cp doc1 dir4/doc2	Se debe mostrar el mensaje de error correspondiente	

2.5.1.10. Comando “mv”: mv [Archivo] [Archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa ruta origen y destino con path completo	mv A./doc1 A./dir1/doc2	Se debe mover el archivo origen en la ruta destino	
Se ingresa solo la ruta origen y destino	mv doc1 dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa solo la ruta origen	mv doc1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un directorio en la ruta origen	mv dir1 dir2	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un archivo en la ruta destino	mv doc1 doc2	Se debe cambiar el nombre del archivo	
Se ingresa un archivo inexistente	mv doc4 doc2	Se debe mostrar el mensaje de error correspondiente	

2.5.1.11. Comando “less”: less [archivo]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa un archivo con path completo	less A./doc1	Se debe mostrar el contenido del archivo en la ruta especificada.	
Se ingresa solo el nombre del archivo	less doc1	Se debe mostrar el contenido del archivo en la ruta actual.	
Se ejecuta el comando sin ingresar el archivo	less	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un directorio	less dir1	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un archivo inexistente	less doc4	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	less dir4/doc1	Se debe mostrar el mensaje de error correspondiente	

2.5.1.12. Comando “write”: write [Archivo] [contenido]

Prueba	Entrada	Resultado Esperado	Resultado obtenido
Se ingresa el nombre con path completo	write A./dir1/pepe contenido	Se debe crear el archivo con el contenido en el directorio especificado.	
Se ingresa el nombre y el contenido	write pepe data	Se debe crear el archivo con el contenido en el directorio actual.	
Se ejecuta el comando sin ningún nombre	write	Se debe mostrar el mensaje de error correspondiente	
Se ingresa solo el nombre	write pepe	Se debe mostrar el mensaje de error correspondiente	
Se ingresa el nombre de un archivo existente	write doc1 data	Se debe mostrar el mensaje de error correspondiente	
Se ingresa un directorio inexistente	write dir4/pepe data	Se debe mostrar el mensaje de error correspondiente	

2.5.2. Grabar las modificaciones del FS en SODIUM y luego levantarlo en Linux

Al desmontar la imagen de Ext2 se sobrescribe el archivo sodium_fat12.img que se crea en la carpeta Build. Para ver las modificaciones desde linux se debe montar la imagen ejecutando:

En SODIUM:
Luego de haber realizado las diferentes operaciones sobre los archivos y directorios ejecutar los siguientes comandos para montar una nueva unidad y poder desmontar la unidad donde se hicieron los cambios.

>> montar FAT12B diskette

>> cd B.

>> desmontar A.

En Linux:

>>mount -o loop -t ext2 sodium_fat12.img /mnt/media

>>ls –la /mnt/media

3. Mejoras Propuestas

3.1. Comandos:

Notamos que los comandos de usuario, actualmente se fijan en los parámetros pasados para su ejecución pero en forma fija. Ejemplo: cp origen destino, siempre el origen es el primer parámetro y destino el segundo. Esto para un comando como cp estaría ok, pero no así para un comando write porque no permitiría escribir texto con espacios. Ejemplo write destino Hola Mundo. Por nuestro lado en la función que es llamada por el comando write, internamente los espacios están contemplados.

También haciendo las pruebas, nos dimos cuenta que la impresión de los mensajes de error lo está realizando cada comando por separado. El VFS solo valida que el número sea menor a 0. Luego cada comando al final tiene un switch validando cada mensaje de error. Nosotros aconsejamos que o bien los mensajes de error estén centralizados en el VFS, o bien los maneje directamente cada modulo del File System por separado. Cualquiera de las 2 opciones estarían bien, aunque aconsejamos que se haga en cada modulo por separado ya que justamente cada modulo sabe los mensajes a imprimir.

Para EXT2 solamente quedo pendiente agregar a los comandos: Copiar carpetas ya que hoy en día el comando de copia (cp), copia archivos, lo mismo pasa con el comando de mover (mv) que mueve archivos y no carpetas. El comando sincronizar con Floppy, notamos que no estaba implementado para ningún otro modulo de FS, pero no obstante no lo vimos como algo importante para esta etapa de SODIUM, ya que al desmontar el Floppy, no se guarda directamente en la misma imagen de EXT2, sino que pisa la imagen de sodium_fat12.img. Esta imagen es la que probamos al desmontar en Linux para verificar que funcione correctamente.

Se modificaron los comandos “find” y “ls”, ya que hacían una comparación incorrecta. Comparaban para el directorio actual y el anterior, es decir el “.” y “..”, como “. “ (punto seguido de 10 espacios) y “..” (punto punto seguido de 10 espacios). Para el caso de EXT2 lo arreglamos para que compare por “.” Y “..”, por lo que quedaría pendiente para FAT arreglar ese bug, para que compare correctamente. Por lo que detectamos esto lo hacían para que quede identado correctamente al momento de listar directorios, en nuestro caso hicimos una función para identarlos y que quede más prolijo.

Asimismo, habría que agregar algún campo más a la estructura de directorios del VFS, ya que en FAT para identificar los directorios escribían la palabra “DIR” en el campo Atributos, pero en el caso de EXT2, este campo se utiliza para los permisos de lectura y escritura, y para diferenciar directorio se busca un campo en dicho atributo la letra “d” (esto es en los comandos “ls” y “find”), ya que el VFS no tiene otra información adicional para distinguir archivos de directorios. Por lo que es recomendable que en VFS se agregue un campo para distinguir si es directorio o archivo u otro.

3.2. Módulos FS (EXT3 y otros)

Cuando empezamos el desarrollo del modulo de EXT2, nos dimos cuenta que muchos de los comandos de FAT (12, 16 o 32) no estaban terminados. También notamos que al hacer las pruebas con dichos módulos, en algunos casos recibíamos errores de excepción de CPU. Estos errores no supimos saber en ese momento si se debió, o a que funcionaron mal o bien había problemas de memoria en SODIUM. Creemos que es el problema de memoria, pero proponemos hacer una revisión de todos los módulos (de FAT y EXT2), no solo probarlos sino también terminarlos.

Una gran mejora, sería el desarrollo del modulo de EXT3 porque creemos que las 2 principales diferencias con EXT2 son sumamente importante. Lo que incorpora EXT3 es el árbol binario balanceado (árbol AVL) y el asignador de bloques de disco Orlov. Recomendamos por el tiempo que puede llevar dicha implementación que sea un trabajo práctico aparte. Asimismo, cuando se realice la implementación de EXT3, se utilice la biblioteca commonExt, ya que la misma contiene cosas en común con los diferentes sistemas de archivos EXT.

3.3. Disco Rígido

Cuando este acoplado y funcionando el modulo de IDE para SODIUM, las modificaciones que deberían hacerse en el modulo de EXT2, serian las 2 funciones de Leer y Escribir (iFnLeerSectorExt y iFnEscribirExt) que actualmente están creadas en RAMFS.

Viendo la implementación del VFS, modulo que permite independizar a cada modulo de FS respecto a SODIUM, creemos que sería de suma importancia crear un modulo de Virtual Hard Disk, para que el mismo sea el nexo entre los distintos módulos de manejo de Disco (IDE, SATA, SCSI, etc.) y los módulos de FS implementados a la fecha.

4. Conclusiones

En base a toda la investigación que estuvimos realizando, tanto sobre el sistema de archivos EXT2, como de SODIUM, y en lo referente al funcionamiento de los mismos, pudimos profundizar a un buen nivel de detalle sobre cómo funciona el FS implementado. Como es su interacción desde los programas de usuario, el Virtual File System o la salida por pantalla, por citar unos ejemplos.

Pudimos comprender el por qué EXT2 es tan robusto como hemos escuchado tantas veces, y a su vez entendimos la complejidad del mismo en comparación con los sistemas de archivos ya implementados (FAT12, FAT16 y FAT32).

A medida que se fue avanzando en el desarrollo del trabajo práctico se observó que al realizar diferentes funciones de usuario, nos encontramos con problemas de excepciones de Memoria, o cuelgues del SODIUM. Luego de chocar varias veces con este tipo de problema, con ayuda de los profesores, nos dimos cuenta que el problema era la cantidad de archivos en Memoria RAM. Vale aclarar que como todavía no está implementado un Disco Duro con IDE o SATA, SODIUM actualmente utiliza lo que llama RAMFS (File System en RAM), utilizando gran parte de la Memoria disponible.

También detectamos que levantando SODIUM en modo paginado era más propenso a quedar el sistema tildado o aparecer excepciones que si utilizábamos modo particionado. Igualmente se notó que si se hacía una cierta cantidad de operaciones consecutivas se llegaba a un punto donde el sistema fallaba independientemente de lo que se ejecutara.

Cuando iniciamos nuestro proyecto, nos dimos cuenta que la implementación de FAT utilizaba funciones en común entre los distintos tipos de FAT. Sabiendo esto, y que además existen distintos tipos de EXT (EXT, EXT2, EXT3, EXT4), nos pareció interesante crear un esquema similar. Es por ello que nosotros creamos un commonExt.c que contiene ciertas funciones genéricas más que nada de obtención de datos. No obstante existen funciones que están dentro del ext2_fs.c pero que tranquilamente podrían generalizarse para los demás tipos de EXT. Esto no lo realizamos en el transcurso del proyecto, debido a que las

estructuras de inodo particularmente son muy distintas y no conocemos en detalle los demás FS, pero igualmente algunas de ellas podrían reutilizarlas para otros FS.