

Universidad ORT Uruguay

Facultad de Ingeniería

Sistemas Operativos Obligatorio

Pablo Benitez - 179924

Fabio Ramirez - 218566

Docente: Ing. Ángel Caffa MSC, MBA

Entregado como requisito de la materia Sistemas Operativos

12 de noviembre de 2019

Resumen ejecutivo

El presente documento se muestra un informe técnico sobre la emulación de un sistema de archivos con un interprete de comandos, muy similar a DOS, programado en C/C++. Se intentan implementar algunas funciones básicas sobre el orden jerárquico directorios, su manejo a nivel de archivos y permisos. No es una copia emulada de DOS, por lo que los comandos representados pueden no coincidir, no existir, o no aparecer en este proyecto.

¿Cómo hicimos esto? Primero procedimos a analizar que hace un sistema de archivos y un SO de consola como DOS. De allí intentamos simular su comportamiento y para ello elegimos las soluciones que creímos más simples, usando las herramientas que eran más fáciles de usar con el lenguaje que seleccionamos.

El resultado del análisis e implementación puede verse en los capítulos que siguen.

Declaraciones de autoría

Nosotros, Pablo Benitez y Fabio Ramirez, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras se realizaba ;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

1. Introduccion	1
2. Objetivos	2
3. Metodología	3
3.1. TAD FS	3
3.2. TAD Usuario	4
3.3. Intérprete de comandos	5
3.4. Encriptación y seguridad	6
4. Corridas	7
4.1. Login incorrecto	7
4.2. Unicidad de usuario	7
4.3. Cantidad de carpetas	8
4.4. Cantidad de líneas de archivo	9
4.5. Mostrar archivo	9
4.6. Atributos de archivo	10
4.7. Carpetas y subcarpetas	11
5. Conclusiones y trabajo futuro	12
5.1. Conclusiones y resultados	12
5.2. Trabajo futuro	12
5.3. Fuentes	13

1. Introduccion

Una parte fundamental de los sistemas Operativos son los File System o Sistemas de Archivos. El principal cometido de este componente es administrar todos los archivos del sistema, manejando las ABM de los mismos, el manejo de espacio en disco, la correcta persistencia e integridad, y su correcto acceso. Los atributos de seguridad sobre estos también deben estar contemplados. Los procesos que se ejecutan generalmente tienen un espacio reservado en memoria, pero si es necesario persistir datos, el file system se encarga también de manejar esto.

La jerarquía de archivos y directorios está organizada en forma de arboles, donde cada componente es un nodo, y cada nodo hijo es un componente que se encuentra dentro de un directorio. Esta estructura está representada en este emulador. El manejo de espacio en disco, de tamaño de archivos y largo de nombres o de cantidad de elementos de un directorio es un atributo funcional de estos sistemas.

Otro tipo de propiedades que maneja son los permisos o atributos sobre archivos. Estos marcan el acceso de manera general o particular a las características del mismo, generando niveles de acceso distintos. Un ejemplo sería: para un archivo de solo lectura, no es lo mismo que intente borrarlo un usuario cualquiera que su autor o propietario.

Generalmente los sistemas operativos tienen que interactuar con algunos dispositivos de almacenamiento masivo, por lo que puede que tengan que trabajar con otros sistemas de archivos. Un caso sería tener acceso a una unidad de CD o trabajar con una memoria flash o una diskettera. Cómo realiza este manejo también es una característica de los sistema de archivos.

En los siguientes capítulos veremos un acercamiento a algunas funciones y su implementación en un formato de emulación. De ahora en más se reconocerá el concepto Sistema de Archivos con las siglas SDA.

2. Objetivos

A continuación se enumeran los principales aspectos que se desean emular de un SDA:

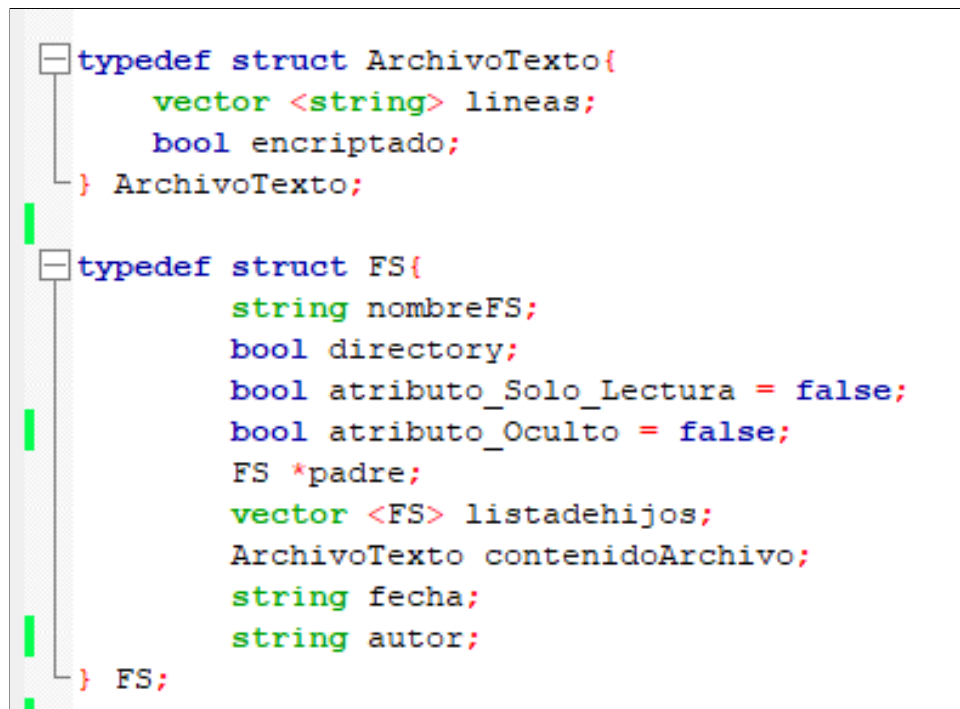
- Estructura en jerarquía - Se pone énfasis en la estructura de directorio con forma de árbol.
- Atributos de componentes de SDA - Manejar los atributos de los componentes del SDA (sean directorios o archivos). Tomamos como base el formato de DOS [1].
- Permisos de usuarios - Con los permisos de usuario se introducen controles sobre las acciones de los mismos.
- Accesos y restricciones de componentes - Usando los atributos y los permisos de usuario, generar una política de seguridad mínima sobre las funciones del sistema.
- Comandos para interacción - Pasamos a implementar una interfaz de usuario minimalista en la que el usuario puede interactuar con el SDA.

Se deja de lado otros aspectos referidos a los SDA en este proyecto, teniendo en cuenta el tamaño de este último, la importancia de la aplicación de la funcionalidad o la complejidad de la solución. Para empezar a comprender el tema recurrimos primero a [2].

3. Metodología

El capítulo actual estará explicando como se diagramó el programa para llevar adelante el cometido presentado más arriba.

3.1. TAD FS



```
typedef struct ArchivoTexto{
    vector <string> lineas;
    bool encriptado;
} ArchivoTexto;

typedef struct FS{
    string nombreFS;
    bool directory;
    bool atributo_Solo_Lectura = false;
    bool atributo_Oculto = false;
    FS *padre;
    vector <FS> listadehijos;
    ArchivoTexto contenidoArchivo;
    string fecha;
    string autor;
} FS;
```

Figura 3.1: TAD FS

Mediante el TAD FS se creó un tipo único de elemento del SDA, con atributos compartidos: nombre, fecha, autor, tipo (Directorio o no), de solo lectura, oculto, de sistema, un listado de hijos para los directorios y un tipo de datos especial para archivos. A todos se les agrega un puntero al directorio padre.

El sistema jerárquico de directorios se implementó mediante un vector de punteros hacia otros elementos del tipo del TAD FS, con lo que pueden ser archivos o directorios. Estos elementos simbolizan los "hijos" de este, es decir, el contenido de la carpeta. El máximo de carpetas o archivos por directorio es de 16. No está definido el máximo de componentes de todo el sistema ya que es solo una simulación.

Para implementar archivos se utilizan los mismos datos que con los Directorios, agregando un vector de string, donde cada elemento simboliza una línea del archivo de texto. Además se agrega una bandera que indica si el archivo está encriptado. El máximo de líneas de archivos es 20 definido como constante. Estos archivos tienen una forma de acceder del tipo secuencial.

Las siguientes características no están implementadas por cuestiones del objetivo de emulación, importancia o complejidad: respaldo del sistema de archivos, administración y optimización del espacio, archivos compartidos, accesos directos (links) y métricas sobre el rendimiento.

Operaciones concretadas:

- Crear
- Borrar
- Abrir Cerrar Leer juntas
- Modificar
- Escribir al final
- Get/Set de atributos
- Renombrar archivo

La implementación de directorio respeta:

- Sistemas de directorios jerárquicos
- Nombres de rutas
- Crear y borrar
- Entrar y salir
- Renombrar

Todo esto siguiendo la guía de operaciones comunes que se presentan en el libro de Tanenbaum[3].

3.2. TAD Usuario

```
typedef struct usuario{
    string nombre;
    string password;
    bool esadmin = false;
} usuario;
```

Figura 3.2: TAD Usuario

Este TAD particularmente fue implementado dentro del main dado su tamaño y la cantidad básica de procedimientos que involucra.

La figura 3.2 muestra claramente que se manejan solo tres atributos sobre los usuarios.

Las operaciones más comunes para usuarios aparecen detalladas entre otras, en el siguiente punto.

3.3. Intérprete de comandos

Teniendo en cuenta las funciones de los SDA que se quieren reflejar, se implementó un interprete de comandos para poder trabajar correctamente. El interprete de comandos solo funciona si existe un usuario logueado, en caso contrario, solo existe la opción de login.

A continuación se enumeran los comandos implementados:

- exit, salir - Salida del sistema.
- logout - Sale de sesión.
- user xxx yyy zzz - crear usuario con nombre xxx pass yyy y tipo zzz.
- listar - Lista usuarios del sistema.
- mkdir DIR1 - crea directorio DIR1.
- mkfile FILE1 - crea archivo FILE1.
- addline FILE1- Agrega linea de texto a un archivo al final.
- rmline FILE1- Elimina una linea especifica de un archivo.
- show FILE - Muestra contenido de archivos.
- dir - Lista archivos de la carpeta actual.
- tree - Muestra árbol de directorios.
- cd FOLDER - Accede a la carpeta FOLDER o con .. sube un nivel.
- hd FILE1 - Deja un archivo FILE1 oculto.
- sl FILE1 - Deja un archivo como solo lectura.
- every - muestra todos los archivos estén o no ocultos.
- crypt FILE1- Encripta el archivo FILE1 o lo descripta.
- attrib FILE1- Muestra atributos de archivo/carpeta FILE1.
- rename FILE1- Renombra archivo/carpeta FILE1.
- cp FILE1- Copia el recurso FILE1 en la carpeta actual a FILE1 Copia.

- `rm FILE1`- Borra el recurso `FILE1` y su contenido.
- `path` - Muestra la ruta actual.
- `passwd` - Cambia la clave del usuario actual.
- `?, help` - Muestra esta ayuda.

3.4. Encriptación y seguridad

En virtud de mantener la seguridad del sistema, se eligieron mantener los permisos sobre archivos, relacionando con los usuarios. Esto es, se distinguen dos tipos de usuarios, normales y administradores. Al ingresar al sistema se tiene un usuario predeterminado 'root' con clave igual a su nombre. Luego de esto pueden crearse varios usuarios para usar en el sistema, con instancia única de login, para realizar todas las funciones que ya fueron mencionadas.

Para encriptar archivo se utiliza el algoritmo XOR [4] con una clave predefinida. Luego de encriptar un archivo solo puede ser desencriptado por el autor del mismo. Solo los usuarios que ocultan un archivo pueden desocultarlo, o un administrador. Lo mismo pasa con la propiedad de solo lectura. Se mantiene la unicidad de nombres de recursos dentro del mismo directorio. Esto es, no se pueden crear dos recursos con el mismo nombre aunque sean de distinto tipo. Esta idea se mantiene para todos los usuarios creados, no pueden haber dos con el mismo login. Como fue visto más arriba, los usuarios pueden cambiar también su password.

4. Corridas

En este capítulo se mostraran varias corridas sobre el programa principal dentro y fuera de los límites del mismo. Dividiremos en secciones cada tipo de control.

4.1. Login incorrecto

Pasaremos a probar primero el acceso con el usuario root y una clave distinta de 'root'.

```
LOGIN
Usuario:
root
Password:
***Credenciales invalidas! Reintente...
Usuario:
_
```

Figura 4.1: Login incorrecto

4.2. Unicidad de usuario

Se probará la creación de dos usuarios con el mismo username.

```
LOGIN
Usuario:
root
Password:
****
Login correcto

Comando :: user juan jp123 admin
Comando :: user juan pepe5 admin

Nombre de usuario ya existe.

Comando :: user root r123 admin

Nombre de usuario ya existe.

Comando :: _
```

Figura 4.2: Unicidad de username

4.3. Cantidad de carpetas

Se prueba la creación de mas del máximo de carpetas (o archivos), borrar una para volver a estar dentro de los parámetros y llegar al máximo nuevamente.

```
LOGIN
Usuario:
root
Password:
****
Login correcto

Comando :: mkdir carpeta1
Comando :: mkdir carpeta2
Comando :: mkdir carpeta3
Comando :: mkdir carpeta4
Comando :: mkdir carpeta5
Comando :: mkdir carpeta6
Comando :: mkdir carpeta7
Comando :: mkdir carpeta8
Comando :: mkdir carpeta9
Comando :: mkdir carpeta10
Comando :: mkdir carpeta11
Comando :: mkdir carpeta12
Comando :: mkdir carpeta13
Comando :: mkdir carpeta14
Comando :: mkdir carpeta15
Comando :: mkdir carpeta16
Comando :: mkdir carpeta17
Directorio lleno.
No existe archivo o directorio
Comando :: rm carpeta9
Comando :: mkdir carpeta17
Comando :: mkdir carpeta18
Directorio lleno.
No existe archivo o directorio
Comando ::
```

Figura 4.3: Maximo de carpetas

4.4. Cantidad de lineas de archivo

Análogamente al caso anterior, se prueba sobrepasar la cantidad de lineas en un archivo.

```
Comando :: addline archivo1
Escriba la linea a agregar: linea15
Comando :: addline archivo1
Escriba la linea a agregar: linea16
Comando :: addline archivo1
Escriba la linea a agregar: linea17
Comando :: addline archivo1
Escriba la linea a agregar: linea18
Comando :: addline archivo1
Escriba la linea a agregar: linea19
Comando :: addline archivo1
Escriba la linea a agregar: linea20
Comando :: addline archivo1
Escriba la linea a agregar: linea21
Archivo lleno.
Comando ::
```

Figura 4.4: Máximo de lineas de archivo

La captura está cortada dado el largo de la salida.

4.5. Mostrar archivo

Luego de agregar lineas a un archivo el contenido del mismo puede mostrarse, este o no oculto o con modo de solo lectura.

```
Comando :: mkfile archivo1
Comando :: addline archivo1
Escriba el texto de la linea: linea1
Comando :: addline archivo1
Escriba el texto de la linea: linea2
Comando :: show archivo1
*****archivo1*****

linea1
linea2

*****Fin de archivo*****
Comando ::
```

Figura 4.5: Mostrar archivo normal

Luego se encripta archivo y se vuelve a mostrar.

```

Comando :: crypt archivo1
Comando :: show archivo1
*****archivo1*****

? #0-p
? #0-s

*****Fin de archivo*****
Comando ::

```

Figura 4.6: Mostrar archivo encriptado

Al usar el mismo comando otra vez, el archivo pasa a desencriptarse.

4.6. Atributos de archivo

Se analizan los atributos y permisos sobre un archivo, en particular en este caso, se procede a mostrar estos atributos, ocultar un archivo y bloquearlo de escritura.

```

Comando :: mkfile archivo1
Comando :: addline archivo1
Escriba el texto de la linea: linea1
Comando :: addline archivo1
Escriba el texto de la linea: linea2
Comando :: hd archivo1
Comando :: addline archivo1
Escriba el texto de la linea: linea3
Comando :: sl archivo1
Comando :: addline archivo1
Escriba el texto de la linea: linea4
Archivo de solo lectura o bloqueado.
Comando :: show archivo1
*****archivo1*****

linea1
linea2
linea3

*****Fin de archivo*****
Comando :: dir
Comando :: mkfile otroarchivo
Comando :: dir
13/11/2019 12:32:28      otroarchivo
Comando :: hd archivo1
Comando :: dir
13/11/2019 12:32:28      archivo1
13/11/2019 12:32:28      otroarchivo
Comando :: attrib archivo1
Atributos de archivo archivo1: H-W-C-
Comando ::

```

Figura 4.7: Solo lectura, Hide y attrib

4.7. Carpetas y subcarpetas

A continuación se muestra la creación de subcarpetas y archivos, el comando `dir`, `path` y `tree`.

```
Comando :: mkdir carpeta1
Comando :: mkdir carpeta2
Comando :: mkdir carpeta3
Comando :: mkfile archivo1
Comando :: cd carpeta3
Comando :: mkfile archivo1
Comando :: mkfile archivo2
Comando :: mkdir carpeta4
Comando :: cd carpeta4
Comando :: mkfile archivo5
Comando :: path
\carpeta3\carpeta4
Comando :: cd ..
Comando :: cd ..
Comando :: dir
9/11/2019 13:6:53 <DIR> carpeta1
9/11/2019 13:6:53 <DIR> carpeta2
9/11/2019 13:6:53 <DIR> carpeta3
9/11/2019 13:6:53      archivo1
Comando :: tree
|carpeta1
|carpeta2
|carpeta3
|__archivo1
|__archivo2
|__carpeta4
|__    archivo5
|archivo1
Comando ::
```

Figura 4.8: `dir` + `path` + `tree`

5. Conclusiones y trabajo futuro

5.1. Conclusiones y resultados

Como resultado de nuestro trabajo, obtuvimos una representación minimal de un sistema de archivos manejado desde una línea de comandos emulada. En este simulador no se implementan aplicaciones del sistema que reproduzcan otros aspectos funcionales, tales como manejo de memoria, de procesos, interrupciones, interacción con hardware, etc.

Concluimos que a pesar de constituir una tema aburrido dentro de los sistemas operativos, puede llegar a resultar para algunos aspectos, trivial, y para otros muy complejo, sobre todo en el manejo de espacio y eficiencia en las operaciones.

5.2. Trabajo futuro

Teniendo en cuenta el avance actual, se pueden sugerir muchos cambios y operaciones nuevas, así como mejoras. Se detallan algunas a nuestro criterio:

- Implementación de accesos directos a archivos o directorios
- Persistencia de archivos en disco.
- Encriptación de archivos directo sobre cada línea y desencriptación según clave proporcionada por usuario.
- Encriptar los datos delicados en memoria previniendo el RAM Hacking.
- Implementar funciones de copia de archivos o directorios en cualquier ubicación permitida.
- Manejo de permisos a nivel de grupos y mejora de permisos a nivel de usuarios.
- Crear funciones de búsqueda e inserción dentro de un archivo en cualquier lugar.
- Simular manejo de espacio en disco y control de cantidad de archivos.
- Mejorar el manejo de errores dentro de la línea de comandos.

5.3. Fuentes

Todas las fuentes de este obligatorio están accesibles en: <https://app.box.com/s/s951oz3q3v0hl6hbz6mr5mai9t7rir1d>. En esta carpeta se encuentran dos archivos con código, y una carpeta con el ejecutable acompañado de una DLL necesaria para su ejecución.

Este texto fue escrito en LaTeX a través de la web de Overleaf. Se puede acceder a las fuentes aquí: <https://www.overleaf.com/read/pbxrkmrqcsty>

Bibliografía

- [1] ELM. Definicion de FAT. [Online]. Available: http://elm-chan.org/docs/fat_e.html
- [2] Wikipedia. Sistema de archivos en Wikipedia. [Online]. Available: https://es.wikipedia.org/wiki/Sistema_de_archivos
- [3] Andrew S. Tanenbaum, *Sistemas operativos modernos*. PRENTICE HALL, 2009.
- [4] kylewbanks. Algoritmo XOR de cifrado. [Online]. Available: <https://kylewbanks.com/blog/Simple-XOR-Encryption-Decryption-in-Cpp>