Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3825 - Sistema de Operación I
Trimestre Enero-Marzo 2025

Simulando la Administración de un Sistema de Archivos UNIX

1. Introducción

El objetivo del proyecto es realizar un simulador de administrador de archivos de un sistema UNIX. Para ello, el programa permite a un usuario operaciones como crear archivos y directorios, listar archivos de un directorio, eliminar archivos y directorios, entre otras.

2. Sobre el Sistema de Archivos

Se quiere simular un sistema de archivo UNIX. Se tiene que los sistemas de archivos UNIX poseen una estructura jerárquica, en forma de árbol. El directorio de más alta jerárquica es "/" y viene a ser la raíz del árbol. Los archivos y directorios que están contenidos en un `directorio base`, se pueden considerar como hijos de ese `directorio base`, en la estructura árbol. Esto quiere decir que el árbol, tiene un número no fijo de ramificaciones. La Figura 1 muestra un ejemplo de la estructura de un sistema de archivos UNIX.

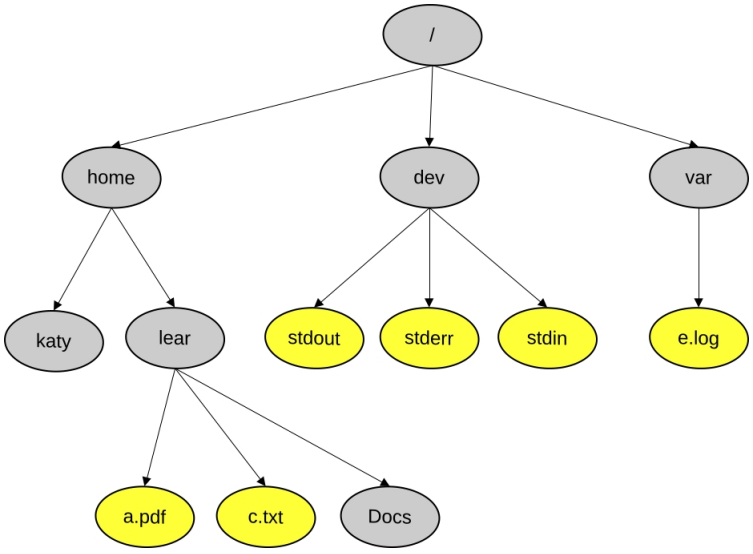

Figura 1: Ejemplo de una jerarquía de archivos UNIX. Los nodos en gris son directorios y los amarillos son archivos.

Por convención, al *directorio actual* en el que se encuentra un usuario, se le denomina "." . Al *directorio inmediatamente* superior al directorio actual, se le denomina "..". Se tiene que la posición de un archivo en sistema de archivo, viene dada por un *camino* o *path*. Hay dos tipos

de caminos, *absoluto* y *relativo*. Se dice que el *camino es absoluto*, si el camino muestra la ruta desde la raíz hasta el archivo o directorio. Por ejemplo, dado el sistema de archivo de la Figura 1, se tiene que el camino absoluto al archivo `stderr`, es `/dev/stderr`. Observe que la separación entre directorios se hace con el símbolo "/". Otro ejemplo, el camino absoluto al directorio `Docs` es `/home/lear/Docs`. Suponga que el usuario se encuentra en un directorio cualquiera X. Se tiene que un *camino es relativo*, si el camino a un archivo o directorio, es dado desde el directorio actual X. Por ejemplo, suponga que el usuario se encuentra en el directorio `lear`, entonces el camino relativo desde ese directorio, hasta el directorio `var` es `../../var`. En un directorio no pueden haber dos archivos o directorios con un mismo nombre. El sistema de archivo es sensible a los caracteres que están en mayúsculas y minúsculas.

3. Representación como Árbol Binario

Como se explica en Cormen et al [1], un árbol con un número arbitrario de ramas en cada nodo, puede representarse como un árbol binario. Usando el mismo principio presentado en [1], se convierte el árbol de la jerarquía del sistema de archivo, en un árbol binario. La idea es que cada nodo tiene un apuntador a su nodo padre, a solo uno de sus hijos y a un solo nodo hermano. Esto es, si un nodo tiene n hijos, él va apuntar a uno de los hijos, luego ese nodo hijo apunta a uno de sus hermanos (el segundo hijo). Luego ese nodo apunta al siguiente hermano, y así sucesivamente hasta que el nodo hijo n, apunta a NULL. De esa manera, se va a tener que todos los nodos, además de apuntar al padre, solo pueden apuntar a dos nodos: a un nodo hijo y a un nodo hermano. Debido a esto, el árbol se transforma en un árbol binario. Para la representación de los nodos en nuestro árbol del sistema de archivos, se va a usar una estructura similar a la presentada en [1]. Se tiene que cada nodo del árbol consta de cuatro elementos: 1. Un apuntador al nodo padre, 2. un apuntador al nodo hijo, 3. un apuntador al nodo hermano, 4. un campo para el nombre, 5. un campo para indicar si el nodo es un archivo o directorio. El la Figura 2 muestra la estructura de un nodo. Se tiene que `Parent` es un apuntador al nodo padre, `Child` es el apuntador al nodo hijo, `Sibling` es un apuntador al nodo hermano, la palabra `File` indica que el nodo es un archivo y la palabra `Dir` que es un directorio. Finalmente, hay un campo `Key` que representa el nombre del archivo o directorio en el sistema de archivos UNIX.

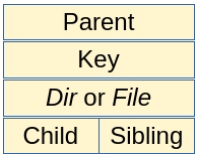

Figura 2: Ilustración de un nodo.

La Figura 3 muestra la representación como un árbol binario del árbol del sistema de archivo UNIX de la Figura 1.

4. Descripción de la Actividad

La idea de la aplicación es proporcionar al usuario de un *intérprete de comandos* UNIX, que inicia en el directorio "/", que le permita ejecutar operaciones relacionadas con el sistema de archivo UNIX. La aplicación debe llamarse `simfs` y se ejecuta en la línea de comandos de la

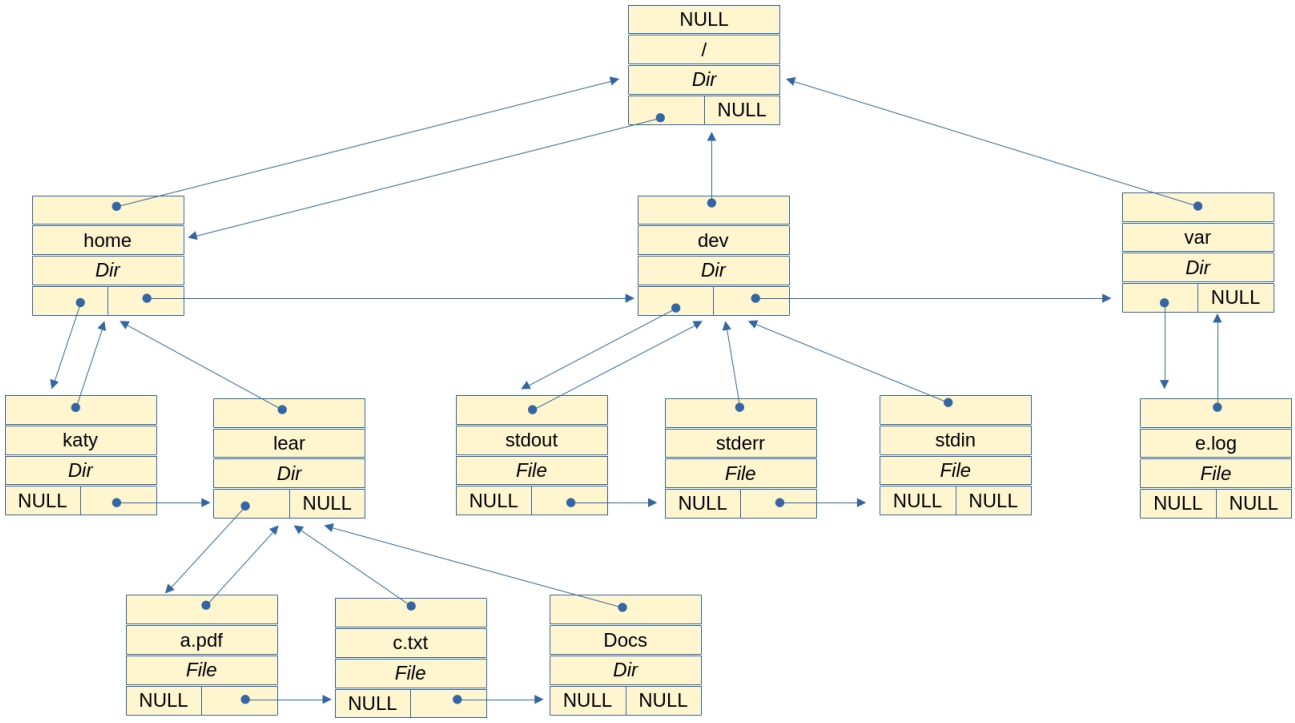

Figura 3: Ilustración de un árbol binario con apuntadores a un hijo y a un hermano.

Figura 4. Este comando tiene como entrada el argumento `archivo_con_unix_fs`, que es opcional, que consiste en un archivo, con formato determinado, que contiene un sistema de archivos de UNIX dado. El intérprete de comandos que debe implementar, se debe comportar como `shell`. Esto es, que debe permitir al usuario ejecutar comandos, y una vez ejecutado, queda a la espera del próximo comando.

```
>./simfs [archivo_con_unix_fs]
```

Figura 4: Descripción del comando que ejecuta la aplicación `simfs`.

Si el comando `simfs`, se ejecuta solo, entonces se debe crear el sistema de archivos solo con el directorio raíz "/", y en ese directorio se debe ubicar al usuario. Si en cambio, la aplicación `simfs` se ejecuta con archivo, entonces debe crear el sistema de archivos a partir de archivo de entrada, y una vez creado el mismo, el interprete de comandos, igual que en el caso anterior, debe ubicar al usuario en el sistema raíz "/". Se tiene que `archivo_con_unix_fs`, corresponde a un archivo de texto con el siguiente formato. Se tiene un archivo o directorio por línea, en donde en primer lugar se presenta el archivo o directorio con su camino completo, en segundo término se presenta separada por tabulación, una letra "D" o "F", si el elemento a agregar al sistema de archivos es un directorio o un archivo respectivamente. A continuación, se presenta el contenido de un archivo de entrada que corresponde al sistema de archivos de la Figura 3.

```
/                D
/home            D
/home/katy       D
/var             D
/var/e.log       F
/home/lear       D
/dev             D
/dev/stdout      F
/home/lear/a.pdf F
/dev/stderr      F
/home/lear/c.txt F
/home/lear/Docs  D
/dev/stdin       F
```

Una vez que se ejecuta `simfs`, se presenta al usuario un intérprete de comando como el de la Figura 5.

```
>
```

Figura 5: Intérprete de la aplicación `simfs`.

Se tiene que una de las propiedades de los archivos y directorios, es que poseen la fecha en que fueron creados.

Los comandos que se pueden ejecutar en el intérprete de comandos son los siguientes: ▪ touch, ▪ rm, ▪ mkdir, ▪ rmdir, ▪ ls, ▪ cd, ▪ pwd, ▪ wrtfs, ▪ help, y ▪ exit

Una *observación importante* es que varios de estos comando reciben como entrada nombres de archivo (< *nombre_archivo* >) o de directorio (< *nombre_directorio* >). Como convención, cuando se indica < *nombre_archivo* > o < *nombre_directorio* >, *a lo que se está refiriendo es al camino absoluto o relativo al archivo o directorio.* Los argumentos de los comandos que se presentan entre menor que y mayor que (<>) son obligatorios, mientras con que en corchetes ([ ]) son opcionales. A continuación se describen los comandos de la aplicación.

4.1. Comando `touch`

Este comando crea un nuevo archivo con el nombre dado. Si el archivo ya ha sido creado o hay un directorio con ese mismo nombre, da un mensaje de error al usuario, indicando que el archivo ya existe. A continuación la sinopsis del comando:

```
>touch <nombre_archivo>
```

4.2. Comando `rm`

Borra un archivo con el nombre dado. Si el archivo no ha sido creado, o si es en lugar de un archivo se intenta borrar un directorio, entonces se da un mensaje de error al usuario. A continuación la sinopsis del comando:

```
>rm <nombre_archivo>
```

4.3. Comando `mkdir`

Crea un directorio con el nombre dado. Si el directorio o archivo con ese nombre ya ha sido creado, entonces se da un mensaje de error al usuario. A continuación la sinopsis del comando:

```
>mkdir <nombre_directorio>
```

4.4. Comando `rmdir`

Borra un directorio vacío con el nombre dado. Si el directorio no existe, o si el directorio no está vacío, o si en lugar de un directorio se intenta borrar un archivo, entonces se da un mensaje de error al usuario. A continuación la sinopsis del comando:

```
>rmdir <nombre_directorio>
```

4.5. Comando `ls`

Este comando muestra los archivos y directorios del directorio dado. Cuando se usa la opción `-l`, se listan los elementos del directorio dado, mostrando los nombres, junto su fecha de creación y si es un archivo o directorio. Los elementos de la fecha de creación a mostrar son la hora, el minuto, el día, el mes y el año que que fue creado el archivo o directorio.

```
>ls [-l] <nombre_directorio>
```

4.6. Comando `cd`

Cambia al usuario de un directorio en el intérprete de comandos, a otro directorio indicado en $< nombre\_directorio >$.

```
>cd <nombre_directorio>
```

4.7. Comando `pwd`

Este comando imprime el camino absoluto hasta el directorio actual en el intérprete de comandos.

```
>pwd <nombre_directorio>
```

4.8. Comando `wrts`

Este comando crea un archivo con la descripción actual del sistema de archivo. La idea es que se recorra en preorden, la estructura de árbol binario que representa al sistema de archivo. Como norma los nodos hijos de un mismo padre (hermanos), debe estar ordenados en orden ascendente a su fecha de creación. Esto es, los archivos y directorios de un directorio dado, deben estar ordenados en el árbol en orden ascendente a su fecha de creación. Por ejemplo, si se aplica este principio al árbol de la Figura 3, para el directorio `lear` se tiene primero se creó el archivo `a.pdf`, luego el archivo `c.txt` y finalmente el directorio `Docs`. A continuación se presenta la sinopsis del comando `wrts`:

```
>wrts <nombre_del_archivo>
```

El formato del archivo de salida del comando imprime los siguientes elementos separados por tabulador: 1. El nombre del archivo o directorio, 2. la fecha de creación, 3. la letra "F" si se trata de un archivo y "D" si es un directorio, 4. el camino absoluto hasta el archivo o directorio incluyéndolo

Por ejemplo, suponiendo que se ejecuta el comando `wrts` al sistema de archivos de la Figura 3, un resultado válido, es un archivo de texto, con el siguiente contenido:

```
/        12:03-12/01/2025    D    /
home     14:00-12/01/2025    D    /home
katy     14:01-12/01/2025    D    /home/katy
lear     15:11-12/01/2025    D    /home/lear
a.pdf    20:44-12/01/2025    F    /home/lear/a.pdf
c.txt    20:50-12/01/2025    F    /home/lear/c.txt
Docs     21:05-12/01/2025    D    /home/lear/Docs
dev      21:06-12/01/2025    D    /dev
stdout   23:08-12/01/2025    F    /dev/stdout
stderr   23:09-12/02/2025    F    /dev/stderr
stdin    23:10-12/03/2025    F    /dev/stdin
var      21:26-12/01/2025    D    /var
e.log    21:26-12/01/2025    F    /var/e.log
```

4.9. Comando `help`

Este comando muestra los comandos disponibles y una breve explicación de cada uno de ellos.

```
>help
```

4.10. Comando `exit`

El comando `exit` termina la ejecución del programa.

```
>exit
```

5. Requerimientos de la Implementación

La estructura que corresponde a un nodo, en el árbol del sistema de archivos que se muestra en la Figura 2, debe ser implementada como una estructura opaca, como la vista en clase. Esto es, en el archivo `node.h`, debe presentar la declaración de la estructura, como lo muestra el Listado 1. En ese archivo también deben aparecer las firmas de los procedimientos y funciones para operar la estructura de un nodo. Entonces, el programador no debe tener acceso a la estructura de un nodo (`nodeStruct`), sino que lo hará por medio de las funciones declaradas para ello. La definición de la estructura `nodeStruct` debe hacerse en el archivo `node.c`, como se muestra en el Listado 2. También las operaciones sobre la estructura `nodeStruct` se implementarán en el mismo archivo. Se recomienda usar la guía de estilo de C que se encuentra en el siguiente enlace: `https://www.cs.umd.edu/~nelson/classes/resources/cstyleguide/`. Debe proporcionar un archivo `Makefile` que compile todo el código, usando los `flags` recomendados en clase, y generar el ejecutable `simfs`.

```
1 enum TYPEFILE {DIR, FILE};
2
3 typedef struct nodeStruct nodeStruct;
4 struct nodeStruct;
```

Listado 1: Declaración de un nodo en el archivo `node.h`

```
1  typedef struct nodeStruct nodeStruct;
2  struct nodeStruct {
3      enum TYPEFILE tf;
4      nodeStruct *parent;
5      nodeStruct *child;
6      nodeStruct *sibling;
7      char *key;
8      ...
9      ...
10 };
```

Listado 2: Definición de una estructura nodo en el archivo `node.c`

6. Condiciones de la entrega

El código del proyecto, el informe y la declaración de autenticidad debidamente firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *Proy1_X_Y.tar.xz*, donde X y Y son los número de carné de los estudiantes. La entrega del archivo *Proy1_X_Y.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 10:00 A.M. del día martes 18 de febrero de 2025.

Referencias

[1] Cormen, T., Leirserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*, 3ra ed. McGraw Hill, 2009.