Introducción

En el presente informe se expone sobre el paradigma lógico aplicado a un simulador de sistema de archivos de un sistema operativo, todo esto bajo el lenguaje de programación de alto nivel “Prolog”.

Este escrito está compuesto por la presentación del problema en el que se comenta los procesos llevados a cabo para dar con una solución a este dicho problema, además de dar algunas instrucciones y sus respectivos ejemplos para terminar finalmente con sus resultados. Finalmente, se da una conclusión de este presente trabajo y así también se entregan los resultados.

Descripción del problema

En el enunciado el problema se solicita diseñar un sistema de archivos, en el que comprende procesos, métodos y reglas que emplea un sistema operativo para la organización y almacenamiento de datos en la memoria no volátil del computador. Dentro de estos archivos se identifica un sistema de archivos que es nuestro primer Tipo de Dato Abstracto (TDA). Luego este sistema de archivos está compuesto de una unidad de almacenamiento (TDA Drive), y dentro de este se almacenan Archivos (TDA File) y carpetas (TDA Folder). Todo esto se requiere implementar bajo el paradigma lógico en conjunto de representar los mencionados TDA anteriormente.

Descripción del Paradigma

El paradigma lógico se puede entender muy bien con la mascota de Prolog “El Buho”, ya que este solo se encarga de verificar si lo que le estamos preguntando (que corresponde a las consultas) es verdad o falso en base a la base de conocimiento que le decimos al Buho que debe saber. En la base de conocimiento le daremos hechos, estos hechos a prolog no le importa si en la realidad es verdad, solo contesta con base a esto dando lugar a muchas posibilidades y con ello dando la posibilidad de programar en prolog.

Este paradigma se basa en esto, bajo esto no existen condicionales ni bucles dado esta condición, además que no existen las funciones. En su reemplazo tenemos las Clausulas que para que una clausula sea verdad se debe cumplir lo que estará adentro de ella, muy similar a lo que cubre una función.

Este paradigma al igual que al paradigma funcional se le indica el ¿Qué? Y no el ¿Cómo? Como en la programación imperativa.

Análisis del Problema

El problema requiere uso estricto de listas para resolver este problema. Quizás viendo este problema se tiende a pensar que hay que usar árboles pero nada lejos de la realidad.
Se busca consultar archivos e información del sistema de archivos lo más rápido posible por lo que todos los archivos y carpetas estarán sobre una misma lista, además realizar las siguientes operaciones bajo el uso estricto de los TDA:

1. Soporte para múltiples unidades físicas y lógicas
2. Soporte para múltiples usuarios
3. Permisos por archivo, carpeta y por usuario
4. Operaciones del tipo:
    1. Compartir (*share*)
    2. Copiar (*copy*)
    3. Mover (*move*)
    4. Eliminar archivos (*del*)
    5. Añadir archivos (*addFile*)
    6. Crear carpetas y subcarpetas, también llamados directorios (*md*)
    7. Eliminar carpetas (*rd*)
    8. Cambiar de directorio (cd)
    9. Renombrar (*ren*)
    10. Buscar dentro del contenido de los archivos (*grep*)
    11. Listar archivos con parámetros tipo /s /a (*dir*)q
    12. Formatear una unidad (*format*)
    13. Encriptar (*encrypt*)
    14. Desencriptar (*decrypt*)
5. Registro de fechas de creación y última modificación de carpetas y archivos
6. Contará con una papelera donde los elementos eliminados quedan alojados de forma temporal.
7. Vaciar papelera de reciclaje
8. Restaurar elementos de papelera de reciclaje

Diseño de la solución

Para el desarrollo de esta se hacen uso de cuatro TDAs:

1.TDA System

Tipo de dato abstracto que representa al sistema de archivos principalmente, que está conformada de esta forma:

Name x Users x Drives x CurrentUser x CurrentDrive x Path x Folders x Trash x CreationDateSystem.

- Name: Corresponde al nombre del sistema de archivos
- Users: Corresponde a una lista de usuarios
- Drives: Corresponde a una lista de unidades fisicas o lógicas
- CurrentUser: Corresponde al usuario que ha iniciado sesión
- CurrentDrive: Corresponde a la unidad que está actualmente siendo usada
- Path: Corresponde al path donde se encuentra el usuario.
- Folders: Corresponde a los archivos que están en la unidad actual que está siendo usada.
- Trash: Papelera de reciclaje
- CreationDateSystem: Fecha de creación del sistema.

2. Tda Drive

Tipo de dato abstracto que representa a una unidad física o lógica, que está conformada de la siguiente forma:

Name x Letter x Folders x Capacity

- Name: Nombre la unidad
- Letter: Letra de la unidad
- Folders: Archivos y carpetas como lista
- Capacity: Capacidad de la unidad.

3. Tda File

Tipo de dato abstracto que representa a un archivo, que está conformado de la siguiente forma:

Name x Content x Path

- Name: Nombre del archivo
- Content: Contenido del archivo
- Path: Ruta donde se encuentra el archivos

4. TDA Folder

Tipo de dato abstracto que representa a una carpeta, está conformado de la siguiente forma:

Name x Password x DecryptnFn x UserCreator x Path

- Name: Nombre de la carpeta
- Password: La contraseña para desencriptar la carpeta.
- DecryptnFn: Función para desencriptar la carpeta
- UserCreator: Creador de la carpeta
- Path: Path de la carpeta

Aspectos de Implemetación

Se utiliza SWI-Prolog version 9.0.4 for x86_64-linux junto con la extensión para VSCode “VSC-Prolog” para el coloreado del código. El proyecto se estructura en los siguientes archivos.

main_21286604_CassoneGonzalez.pl

Archivo donde se encuentran los requerimientos del proyecto.

file_21286604_CassoneGonzalez.pl

Se encuentra el TDA File

folder_21286604_CassoneGonzalez.pl

Se encuentra el TDA Folder

system_21286604_CassoneGonzalez.pl

Se encuentra el TDA System

drive_21286604_CassoneGonzalez.pl

Se encuentra el TDA Drive

script_21286604_CassoneGonzalez.pl

Archivo de código que contiene ejemplos de los requerimientos funcionales.

Instrucciones de Uso

Para crear y utilizar los TDAs respectivos hay que respetar el dominio de cada predicado especificado en el archivo de código y además en el presente informe.
Ejemplos de creación de TDAs:

Tda System - constructor

system("newSystem", System).

Se unificaran Name, Users, Drives, CurrentUser, CurrentDrive, Path, Folders, Trash, y Timestamp en una lista con nombre “newSystem”

Tda File – constructor

file("foo1.txt", "hello world", Pepe).

Se crea un archivo de nombre “foo1.txt” con contenido “hello world, se almacena en Pepe.

Tda Folder- constructor

makeFolder(“Folder1”, “”, “”, “user1, “/”, Folder).

Se crea carpeta1 que le pertenece a user1.

Tda Drive – constructor

makeDrive(“Unidad1”, “C”, [], “1000”).

Se crea una unidad llamada Unidad1 con letra C de 1000MiB.

Resultados y autoevaluación

Los resultados no fueron los esperados debido que faltaron algunos requerimientos que están expresados en la autoevaluación por falta de tiempo.

Conclusiones

Al utilizar el paradigma lógico en comparación al paradigma funcional es mucho más complejo ver la recursión o hacer comparaciones. No se ve tan a simple vista y puede resultar muy tedioso programar de esta manera.
La solución se ve desarrollada con inconvenientes dado a que no se terminaron de implementar todos los requerimientos.