INF-253 Lenguajes de Programación
Tarea 2: C

19 de abril de 2023

1. Liaf

Al programar, tenemos mucha libertad a la hora de definir como hacer las cosas, y por lo mismo se vuelve muy fácil ser desordenado. Para evitar esto, solemos establecer reglas a seguir en un proyecto y construimos herramientas que nos ayuden a seguir estas reglas. Ya hemos construido una herramienta que nos permite forzar una estructura en nuestro código (**Formatter**), pero proyectos grandes están formados de múltiples archivos y carpetas. Es por esto que se te ocurrió la gran idea de hacer tu propio sistema de archivos: Liaf (**L**iaf **I**s **A** **F**ile **S**ystem) el cual permitira estructurar proyectos en un arbol de directorios.

2. Definición

Para esta tarea usted deberá crear un programa en C capaz de administrar archivos y carpetas en base a las instrucciones indicadas por consola. Para lograr este objetivo, deberá crear una estructura de datos de tipo árbol, capaz de almacenar tanto carpetas como archivos haciendo uso de memoria dinámica, punteros a void y punteros a función.

2.1. Instrucciones a recibir

Su programa deberá leer desde la consola una serie de instrucciones. Estas instrucciones pueden ser las siguientes:

- `mkdir <nombre_directorio>`: Crea un directorio con nombre `<nombre_directorio>` en el directorio actual.
- `cd <nombre_directorio>`: Selecciona al directorio `<nombre_directorio>` en el directorio actual como el nuevo directorio actual. Si `<nombre_directorio>` es `..` entonces el nuevo directorio actual sera el padre del directorio actual.
- `touch <nombre_archivo>`: Crea un archivo con nombre `<nombre_archivo>` en el directorio actual.
- `write <nombre_archivo> "<contenido>"`: Escribe `<contenido>` como el contenido del archivo `<nombre_archivo>` del directorio actual.
- `cat <nombre_archivo>`: Imprime el contenido del archivo `<nombre_archivo>` del directorio actual.
- `ls`: Imprime los nombres de todos archivos y directorios del directorio actual.

- `ls <nombre_directorio>`: Imprime los nombres de todos archivos y directorios del directorio `<nombre_directorio>` en el directorio actual. Si `<nombre_directorio>` es `.` se comporta de forma idéntica a ejecutar `ls`.
- `mapdir <instrucción>`: Ejecuta `<instrucción>` en cada subdirectorio del directorio actual. `instrucción` puede ser una de las siguientes instrucciones:
  - `touch <nombre_archivo>`
  - `ls <nombre_directorio>`
  - `mkdir <nombre_directorio>`

Notas:

- Los nombres de directorios y los nombres de archivos solo pueden estar formados por letras, números y puntos.

2.2. Estructura de datos

Los directorios y archivos deberán ser almacenados en una estructura de datos de la siguiente forma:

```
typedef struct Nodo {
    struct Nodo* padre;
    char tipo[64];
    void* contenido;
} Nodo;

typedef struct {
    int largo_actual;
    int largo_maximo;
    Nodo* arreglo;
} Lista;

typedef struct {
    char nombre[128];
    Lista* hijos;
} Directorio;

typedef struct {
    char nombre[128];
    char contenido[256];
} Archivo;
```

- Nodo: Estructura que permite guardar un Directorio o un Archivo. `padre` es un puntero al nodo padre en el árbol de directorios. `tipo` es un string que puede ser `Directorio` o `Archivo` e indica el tipo de dato de `contenido`.
- Lista: Corresponde a una lista de tipo Arreglo dinámico.
- Directorio: La lista `hijos` es utilizada para almacenar los subdirectorios y archivos contenidos en un directorio.
- Archivo: El string `contenido` corresponde a lo que esta escrito en el archivo (se imprime por pantalla con operaciones `cat` y se sobre escribe con operaciones `write`).

2.3. Funciones

Además, se deben implementar las siguientes funciones para tratar con la estructura de datos anteriormente descrita y realizar las operaciones solicitadas por el usuario:

```
// Crea una nueva lista.
Lista* crear_lista(int largo_maximo_inicial);

// Agrega el nodo en la posicion lista.largo_actual de lista.arreglo.
// Si lista.largo_actual >= lista.largo_maximo, entonces se duplica
// el tamaño del arreglo y se actualiza lista.largo_maximo.
void insertar_lista(Lista* lista, Nodo* nodo);

// Retorna el hijo del directorio actual de un nombre dado.
Nodo* buscar_directorio(Directorio* actual, char* nombre);
Nodo* buscar_archivo(Directorio* actual, char* nombre);

// Crea un nuevo Nodo
Nodo* crear_nodo(Nodo* padre, char* tipo, char* nombre);

// Crea un directorio y lo agrega como hijo del actual.
void mkdir(Nodo* actual, char* nombre_directorio);

// Crea un archivo y lo agrega como hijo del directorio actual.
void touch(Nodo* actual, char* nombre_archivo);

// Busca un archvio y cambia su contenido.
void write(Nodo* actual, char* nombre_archivo, char* contenido);

// Busca un archivo e imprime su contenido.
void cat(Nodo* actual, char* nombre_archivo);

// Imprime todos los hijos del directorio actual o del seleccionado.
void ls(Nodo* actual)
void ls_dir(Nodo* actual, char* nombre_directorio);

// Aplica la instruccion a todos los subdirectorios.
void mapdir(
    Nodo* actual,
    void (*insturccion)(Nodo*, char*),
    char* parametro_instruccion
);
```

3. Ejemplo

En el siguiente ejemplo se utiliza > para denotar las instrucciones ingresadas por el usuario.

```
> touch README
> write README Contenido del readme de ejemplo
> cat README
Contenido del readme de ejemplo
> mkdir src
> ls
./src
README
> cd src
> mkdir tests
> mkdir data
> mkdir models
> mkdir utils
> touch index.js
> mapdir mkdir subdir
> mapdir touch archivo
> ls
./tests
./data
./models
./utils
index.js
mapdir ls .
./subdir
archivo
./subdir
archivo
./subdir
archivo
./subdir
archivo
cd ..
ls
./src
README
```

4. Sobre Entrega

- Se deberá entregar un programa con los siguientes archivos:
  - `arbol.c`: Contiene la implementación de las funciones de la estructura de datos.
  - `arbol.h`: Contiene la definición de los struct y funciones de la estructura de datos.
  - `main.c`: Contiene el código encargado de leer las instrucciones, generar la estructura de datos, y escribir por consola los resultados de las instrucciones.
  - `makefile`: Contiene el código para compilar su programa utilizando `make`.
- Los ayudantes correctores pueden realizar descuentos en caso de que el código se encuentre muy desordenado.
- Las funciones implementadas deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**

```
/*
Descripcion de la funcion

    Parametros:
            a (int): Descripcion del parametro a
            b (int): Descripcion del parametro b

    Retorno:
            c (str): Descripcion del parametro c
*/
```

- Debe estar presente el archivo MAKEFILE para que se efectué la revisión, este debe compilar **TODOS** los archivos.
- **Si el makefile no está bien realizado, la tarea no se revisará.**
- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea2LP_RolAlumno.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa. De no incluir README se realizara un descuento.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **Martes 25 de abril**.
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- **Solo se contestaran dudas realizadas en AULA y que se realicen al menos 48 horas antes de la fecha de entrega.**
- Se utilizará Valgrind para detectar los leak de memoria.

5. Calificación

5.1. Entrega Mínima

Para la calificación de su tarea, debe realizar una entrega con requerimiento mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir. La entrega mínima debe ser capaz de realizar las siguientes instrucciones (esto incluye I/O por consola):

- `touch <nombre_archivo>`
- `ls`

Para esto se deben utilizar todos los structs indicados y tener implementado como mínimo las funciones:

- `crear_lista`
- `insertar_lista`
- `crear_nodo`
- `touch`
- `ls` (solo directorio actual)

5.2. Entrega

- Funciones (**60 pts**)
  - `buscar_directorio` y `buscar_archivo` (**5 pts**)
  - `mkdir` (**5 pts**)
  - `write` (**10 pts**)
  - `cat` (**10 pts**)
  - `ls_dir` (**5 pts**)
  - `mapdir` (**25 pts**)
- I/O por consola (**10 pts**)

Se asignará puntaje parcial por funcionamiento parcialmente correcto.

5.3. Descuentos

- Falta de comentarios (-10 pts c/u MAX 30 pts)
- Código no compila (-100 pts)
- Warning (c/u -5 pts)
- Memory leaks (entre -5 y -20 pts dependiendo de cuanta memoria se pierda)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (máximo -20 pts)
- Día de atraso (-20 pts por día, -10 pts dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)