

Proyecto dwimsh – Do What I Mean (DWIM) Shell

Semana 10

Nombre del estudiante:

Daniel Isaac Juárez Funes

Número de cuenta:

12141153

Sede de estudio:

UNITEC TGU

Docente:

Ing. Román Arturo Pineda

Asignatura y Sección:

CCC401 – Sistemas Operativos I, Sec.890

Fecha de entrega:

Lunes 23 de septiembre de 2024

Índice

Descripción y Funcionamiento del Proyecto	3
Descripción	3
Funcionamiento.....	3
Uso del Proyecto	4
Descripción de Rutinas y Procedimientos	7
LoadCommands	7
IsCommandInTable.....	7
FreeCommandsMemory	7
TokenizeUserInput	7
PrintTokens.....	7
DistanciaHamming	7
Comparador.....	8
SonAnagramas.....	8
JoinUserRecommendation	8
ListCommandsTable	8
DeleteDuplicatedRecommendations	8
Comentarios de Implementación	9
Listado del Programa	10

Descripción y Funcionamiento del Proyecto

Descripción

El proyecto “Do What I Mean Shell” consiste en el desarrollo de un shell interactivo bajo el concepto “dwimsh”, que se ejecuta en un entorno Minix3. El objetivo principal del programa es proporcionar una interfaz de línea de comandos que permita a los usuarios ejecutarlos comandos encontrados en el directorio especificado en el entorno y también procesar y mostrar recomendaciones de comandos similares en caso de errores de escritura.

Funcionamiento

El funcionamiento del programa se basa en el manejo del flujo de la ejecución desde que el usuario ingresa un comando. Al iniciar, el programa carga todos los comandos disponibles en el directorio indicado, almacenándolos en una tabla en memoria. Esto permite que el shell acceda de una manera óptima a los comandos, los valide con el input del usuario y los compare con los disponibles en el sistema.

El input del usuario se procesa dividiendo el comando en tokens, utilizando el espacio como delimitador. Estos tokens se utilizan tanto para la validación como para la ejecución de comandos. Si un comando ingresado existe en la tabla de comandos cargados, el programa crea un proceso hijo y luego ejecuta el comando. El shell espera a que el proceso hijo termine antes de aceptar un nuevo comando para poder dar fin a la ejecución completa del primero de ellos.

En caso de que el usuario ingrese un comando no reconocido, el shell genera recomendaciones utilizando dos métodos, la **Distancia de Hamming** y la **Detección de Anagramas**. La Distancia de Hamming corrobora que la longitud de los comandos a comparar sea igual y compara los caracteres del comando ingresado con los disponibles uno a uno. La Distancia de Hamming incrementa si los caracteres son diferentes, y si la diferencia total entre caracteres es suficientemente pequeña, se sugiere el comando similar. La Detección de Anagramas corrobora que la longitud de los comandos a comparar sea igual, ordena los caracteres del comando alfabéticamente utilizando bubbleSort y luego analiza los comandos disponibles en busca de anagramas (o igualdad, ya que este se basa en encontrar las mismas letras, pero en diferente posición) del comando ingresado, sugiriendo aquellos que coincidan tras ordenar sus caracteres. Al implementar estos métodos, se le dio prioridad y se tomó como parámetro principal a la longitud del comando a la hora de filtrar las recomendaciones.

Una vez el usuario ingresó un comando no encontrado y el programa recopiló las recomendaciones, el shell ofrece una interacción sencilla, donde el usuario puede aceptar o rechazar las recomendaciones de comandos. Si se acepta una sugerencia, el shell ejecuta el comando recomendado con los mismos argumentos ingresados por el usuario. En caso de ser rechazada, pasa al próximo comando o al final del ciclo de ejecución, en su defecto.

El proyecto incluye funciones para liberar la memoria utilizada, asegurando que no haya fugas de memoria después de la ejecución del programa.

Uso del Proyecto

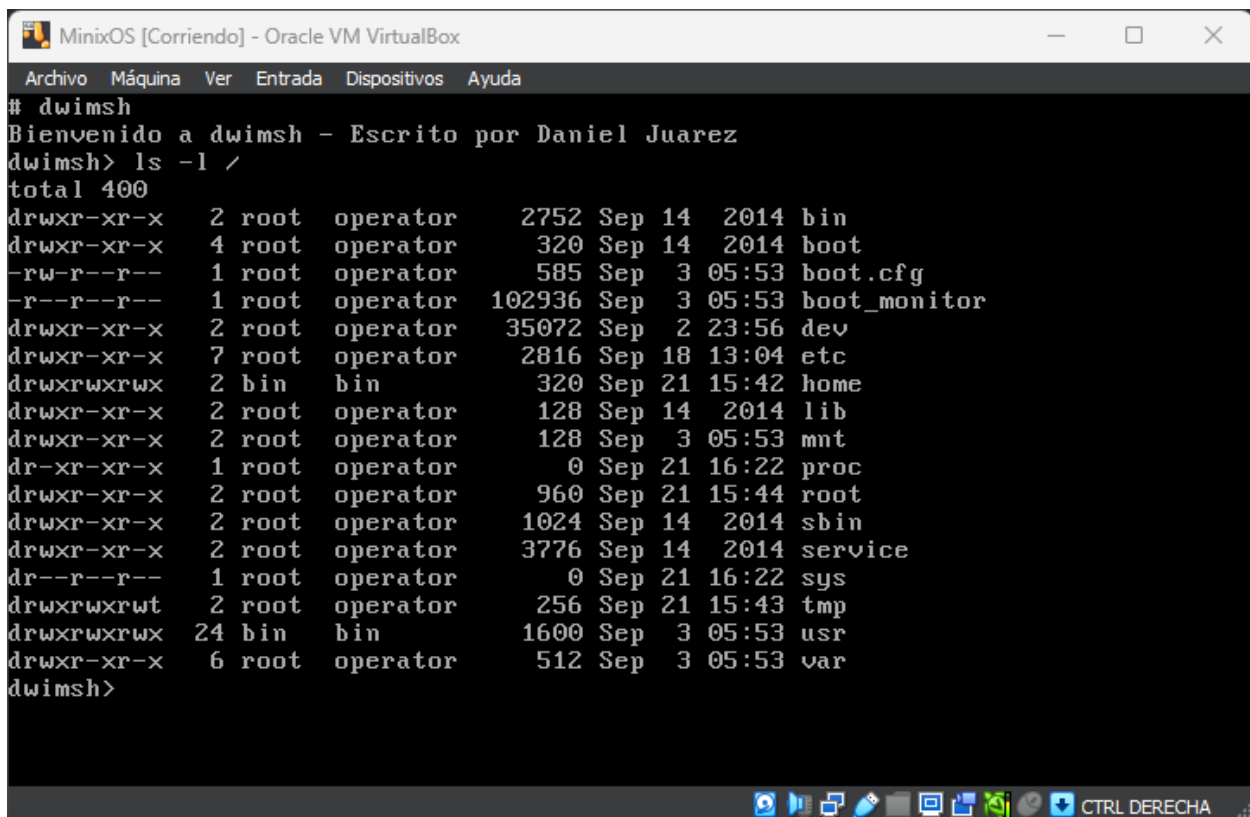
Una vez el archivo compilado fue copiado al directorio para almacenar los programas ejecutables de Minix y también se añadió el correspondiente permiso de ejecución, el programa está listo para ser ejecutado desde cualquier ubicación en Minix.

Este programa es un shell que permite ejecutar los comandos del sistema simulando una terminal. Al iniciar el programa con el comando *dwimsh*, el programa se inicia dándonos la bienvenida al shell interactivo.



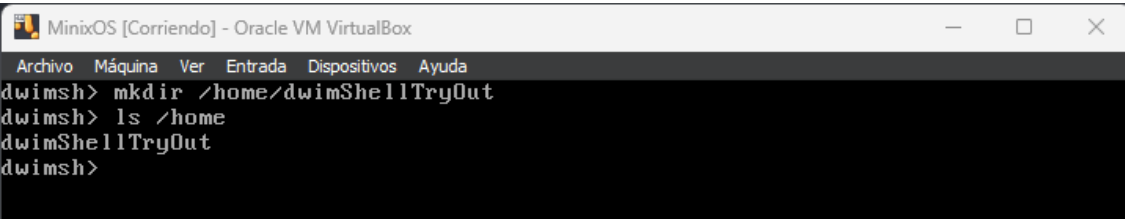
```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh>
```

En este paso, podemos comenzar a ejecutar comandos como los haríamos normalmente en cualquier terminal. Siempre tendremos el mensaje *dwimsh>* previo a nuestro input, mostrándonos que seguimos dentro del programa. A continuación, se muestra un input exitoso por parte del usuario, listando con detalles la carpeta raíz del sistema



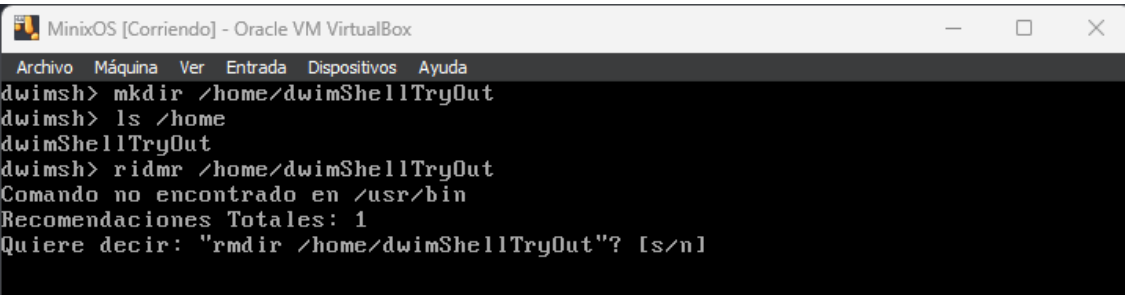
```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh> ls -l /
total 400
drwxr-xr-x  2 root  operator   2752 Sep 14  2014 bin
drwxr-xr-x  4 root  operator    320 Sep 14  2014 boot
-rw-r--r--  1 root  operator    585 Sep  3 05:53 boot.cfg
-r--r--r--  1 root  operator 102936 Sep  3 05:53 boot_monitor
drwxr-xr-x  2 root  operator  35072 Sep  2 23:56 dev
drwxr-xr-x  7 root  operator   2816 Sep 18 13:04 etc
drwxrwxrwx  2 bin   bin       320 Sep 21 15:42 home
drwxr-xr-x  2 root  operator   128 Sep 14  2014 lib
drwxr-xr-x  2 root  operator   128 Sep  3 05:53 mnt
dr-xr-xr-x  1 root  operator    0 Sep 21 16:22 proc
drwxr-xr-x  2 root  operator   960 Sep 21 15:44 root
drwxr-xr-x  2 root  operator  1024 Sep 14  2014 sbin
drwxr-xr-x  2 root  operator  3776 Sep 14  2014 service
dr--r--r--  1 root  operator    0 Sep 21 16:22 sys
drwxrwxrwt  2 root  operator   256 Sep 21 15:43 tmp
drwxrwxrwx 24 bin   bin      1600 Sep  3 05:53 usr
drwxr-xr-x  6 root  operator   512 Sep  3 05:53 var
dwimsh>
```

Acá podemos ver la creación de un nuevo directorio en la carpeta home.



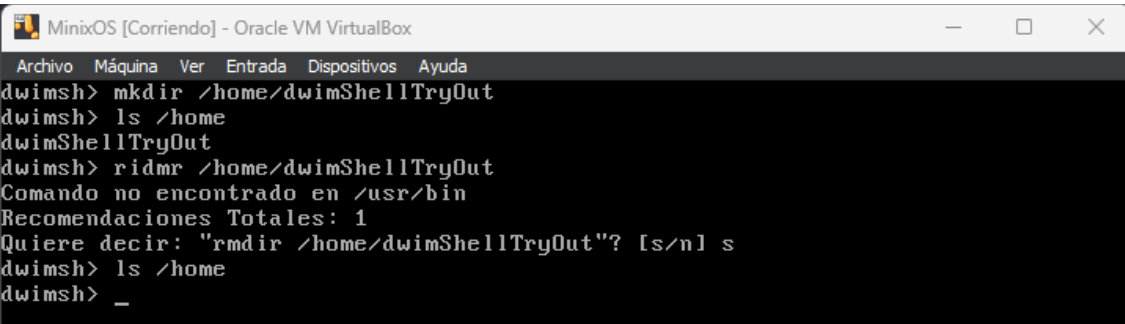
```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
dwimsh> mkdir /home/dwimShellTryOut
dwimsh> ls /home
dwimShellTryOut
dwimsh>
```

Se ingresó el comando *ridmr* por error, la intención original del usuario era utilizar el comando *rmdir* utilizado para borrar directorios. El programa advierte al usuario que no se pudo ejecutar el comando ya que este no existe, pero se encontraron sugerencias. El usuario es presentado con las sugerencias encontradas.



```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
dwimsh> mkdir /home/dwimShellTryOut
dwimsh> ls /home
dwimShellTryOut
dwimsh> ridmr /home/dwimShellTryOut
Comando no encontrado en /usr/bin
Recomendaciones Totales: 1
Quiere decir: "rmdir /home/dwimShellTryOut"? [s/n]
```

El programa no actúa sin la confirmación del usuario. Se le presenta el prompt para decidir si ese era el comando que deseaba ejecutar originalmente. Al decidir que si e ingresar esta opción, el comando mostrado en pantalla se ejecutará, en conjunto con los flags y parámetros propuestos previamente, con éxito.



```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
dwimsh> mkdir /home/dwimShellTryOut
dwimsh> ls /home
dwimShellTryOut
dwimsh> ridmr /home/dwimShellTryOut
Comando no encontrado en /usr/bin
Recomendaciones Totales: 1
Quiere decir: "rmdir /home/dwimShellTryOut"? [s/n] s
dwimsh> ls /home
dwimsh> _
```

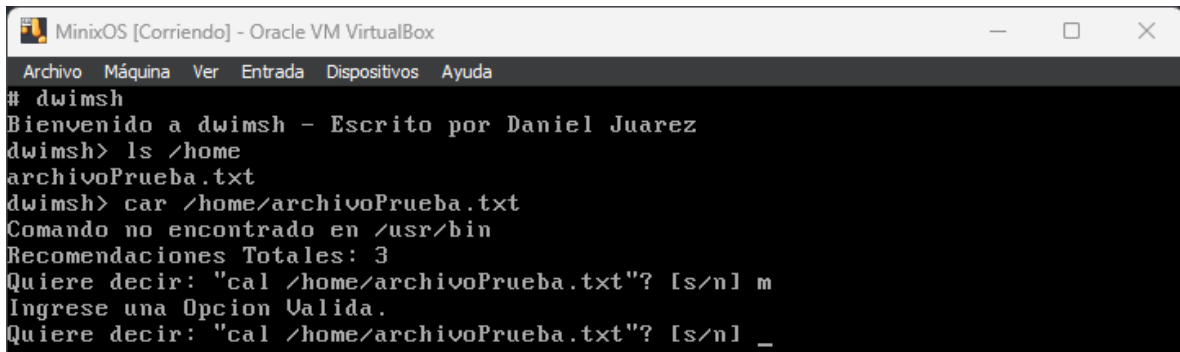
Para terminar la ejecución del programa, ingresamos el comando *exit*.



```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh> exit
#
```

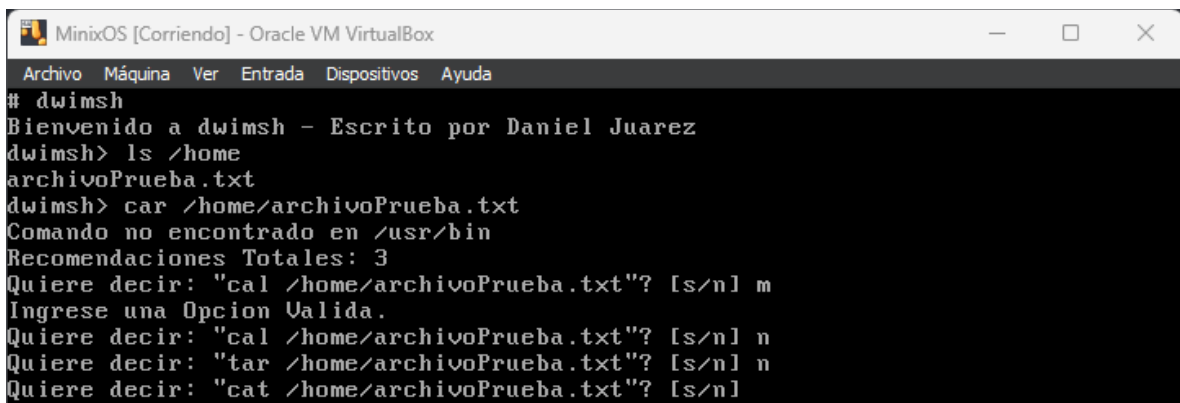
A continuación, trataremos de leer el contenido de un archivo de texto utilizando el comando *cat*. Ubicamos el archivo “archivoPrueba.txt” en el directorio home.

Se ingresó el comando *car* por error, la intención original del usuario era utilizar el comando *cat* utilizado para imprimir el contenido de un archivo. El programa advierte al usuario que no se pudo ejecutar el comando ya que este no existe, pero se encontraron sugerencias. El usuario es presentado con la primera sugerencia, pero el usuario tuvo un misclick, el programa vuelve a mostrar la sugerencia ya que no fue una acción clara por parte del usuario.

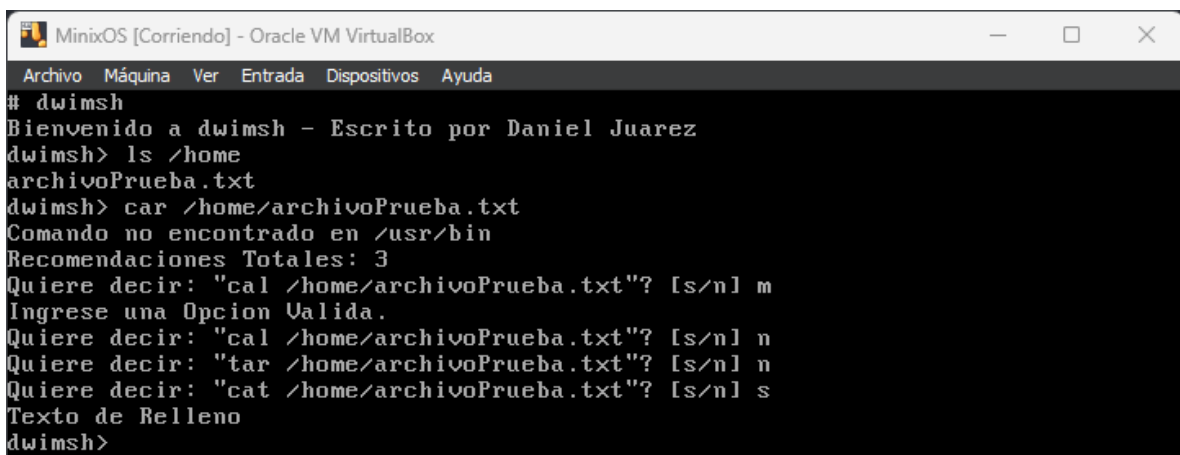


```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh> ls /home
archivoPrueba.txt
dwimsh> car /home/archivoPrueba.txt
Comando no encontrado en /usr/bin
Recomendaciones Totales: 3
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] m
Ingresa una Opcion Valida.
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] _
```

El usuario decide no utilizar las primeras dos de las tres sugerencias presentadas. Llegando a la tercera y última, que es el comando originalmente deseado y ejecutado originalmente.



```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh> ls /home
archivoPrueba.txt
dwimsh> car /home/archivoPrueba.txt
Comando no encontrado en /usr/bin
Recomendaciones Totales: 3
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] m
Ingresa una Opcion Valida.
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] n
Quiere decir: "tar /home/archivoPrueba.txt"? [s/n] n
Quiere decir: "cat /home/archivoPrueba.txt"? [s/n]
```



```
MinixOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
# dwimsh
Bienvenido a dwimsh - Escrito por Daniel Juarez
dwimsh> ls /home
archivoPrueba.txt
dwimsh> car /home/archivoPrueba.txt
Comando no encontrado en /usr/bin
Recomendaciones Totales: 3
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] m
Ingresa una Opcion Valida.
Quiere decir: "cal /home/archivoPrueba.txt"? [s/n] n
Quiere decir: "tar /home/archivoPrueba.txt"? [s/n] n
Quiere decir: "cat /home/archivoPrueba.txt"? [s/n] s
Texto de Relleno
dwimsh>
```

Descripción de Rutinas y Procedimientos

LoadCommands

Carga en memoria los nombres de los comandos disponibles en */usr/bin*. Lee todos los archivos regulares que contienen y por cada comando encontrado, se guarda en el arreglo *cmdTable*. La función utiliza las estructuras *DIR* para manipular directorios y *dirent* para los archivos. Si no se puede acceder a los directorios mencionados, la función muestra un mensaje de error y termina la ejecución.

IsCommandInTable

Verifica si el comando ingresado por el usuario existe en la tabla de comandos cargada. Compara el comando ingresado con los nombres almacenados en *cmdTable*. Si se encuentra una coincidencia, retorna 1. En caso contrario, retorna 0. Esto permite al programa saber si puede ejecutar directamente el comando o si debe buscar recomendaciones.

FreeCommandsMemory

Libera la memoria asignada a la tabla de comandos *cmdTable*, asegurando que no queden memory leaks después de finalizar el programa.

TokenizeUserInput

Convierte una línea de comando ingresada por el usuario en un conjunto de tokens. Usa *strtok* para dividir el input en palabras separadas por espacios. Cada token generado es almacenado en el arreglo *tokens*. Esto permite que el programa ubique el comando, los flags y los argumentos como elementos individuales para trabajar las sugerencias y más adelante unir el resto del input.

PrintTokens

Itera sobre el arreglo de tokens e imprime en pantalla los tokens generados por la función *TokenizeUserInput*.

DistanciaHamming

Calcula la distancia Hamming entre dos cadenas y sugiere comandos basados en esa distancia. Primero verifica si las dos cadenas tienen la misma longitud. Si es así, compara cada carácter en ambas cadenas. Por cada diferencia encontrada, se suma a la distancia Hamming. Si la distancia es suficientemente baja, la cadena que proviene de *cmdTable* se toma como una recomendación y se almacena en el arreglo *recommendations*.

Comparador

Es utilizado en el algoritmo *qsort* para ordenar cadenas de caracteres. Compara dos caracteres y retorna la diferencia entre ellos.

SonAnagramas

Verifica si dos cadenas de caracteres son anagramas entre sí. Si las dos cadenas tienen la misma longitud, ordena sus caracteres usando *qsort* y *Comparador* y las compara. Si ambas son iguales después de ser ordenadas, se toman como anagramas, y la cadena que proviene de *cmdTable* se toma como una recomendación y se almacena en el arreglo *recommendations*.

JoinUserRecommendation

Combina el comando recomendación elegido por el usuario con los argumentos de los tokens del comando original, y crea un nuevo comando completo listo para ser ejecutado.

ListCommandsTable

Itera sobre *cmdTable* e imprime cada comando en la consola, útil para depurar o listar los comandos disponibles.

DeleteDuplicatedReccomendations

Elimina comandos duplicados de la lista de recomendaciones. Compara las recomendaciones y elimina aquellas que se repiten. Para ello, utiliza un ciclo que compara cada recomendación con las demás y, si encuentra duplicados, ajusta el arreglo *recommendations* eliminando las entradas duplicadas.

Comentarios de Implementación

Previo a la implementación del proyecto, se configuró el entorno MINIX3 en una máquina virtual, sin embargo, la imagen de MINIX3 utilizada presentó varias limitaciones. Entre ellas, no era posible usar las funciones de copiar y pegar entre el host y la máquina virtual, lo cual complicaba el manejo eficiente del código. Tampoco contaba con la funcionalidad de archivos compartidos entre el host y la máquina virtual, lo que forzó a buscar otras alternativas. Se intentaron instalar varias herramientas, pero no estaban disponibles en los repositorios de MINIX3.

Después de investigar diversas soluciones sin éxito, opté por una estrategia alternativa: usar WSL como intermediario para transferir archivos desde el sistema operativo Windows hacia Linux. Una vez en Linux, se utilizó el protocolo FTP para enviar y obtener los archivos al y del sistema MINIX. Dentro de MINIX, para poder compilar el código, se utilizó el compilador Clang.

Cubriendo tecnicidades del proyecto, uno de los principales retos fue encontrar un rango adecuado de distancia Hamming para asegurar que las sugerencias fueran útiles, pero no excesivas. Se optó por ajustar tomar como parámetro principal la longitud de los comandos y el rango se decidió dependiendo de la longitud de los comandos.

Minix3 demostró ser un entorno minimalista ya que presentó muchas limitaciones en cuanto a herramientas disponibles y manejo de memoria. También, varios de los comandos que conocemos normalmente por CMDs como el de Windows o Linux, se comportaban diferente o directamente no estaban.

También para hacer que la interfaz del shell fuera más amigable, se decidió implementar un sistema de interacción visual sencilla, con instrucciones al usuario y mensajes de confirmación de comandos sugeridos intuitivos para el usuario. En un entorno con mayor soporte gráfico se podrían implementar mejoras visuales como colores para diferenciar errores y sugerencias, o un sistema de autocompletado que muestre las sugerencias en tiempo real.

Finalmente, atacando el atractivo más grande del proyecto, como una mejora futura se podrían agregar algoritmos más avanzados de uso común y guardando métricas del uso del programa basadas en el historial de comandos del usuario para hacer que las sugerencias sean más relevantes y precisas.

Listado del Programa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6 #include <dirent.h>
```

```
1 #define MAX_CMD_LENGTH 256
2 #define MAX_CMDS 1024
3
4 char *cmdTable[MAX_CMDS];
5 int cmdCount = 0;
```

Includes para librerías utilizadas, consts definidas y variables globales

```
1 // Cargar comandos de /usr/bin a memoria
2 void LoadCommands() {
3     DIR *dir;
4     struct dirent *entry;
5
6     dir = opendir("/usr/bin");
7     if (dir != NULL) {
8         while ((entry = readdir(dir)) != NULL && cmdCount < MAX_CMDS) {
9             if (entry->d_type == DT_REG || entry->d_type == DT_LNK) {
10                 cmdTable[cmdCount] = strdup(entry->d_name);
11                 cmdCount++;
12             }
13         }
14         closedir(dir);
15     } else {
16         perror("No se pudo abrir el directorio /usr/bin");
17         exit(EXIT_FAILURE);
18     }
19
20     dir = opendir("/bin");
21     if (dir != NULL) {
22         while ((entry = readdir(dir)) != NULL && cmdCount < MAX_CMDS) {
23             if (entry->d_type == DT_REG || entry->d_type == DT_LNK) {
24                 cmdTable[cmdCount] = strdup(entry->d_name);
25                 cmdCount++;
26             }
27         }
28         closedir(dir);
29     } else {
30         perror("No se pudo abrir el directorio /bin");
31         exit(EXIT_FAILURE);
32     }
33 }
```

Procedimiento LoadCommands

```
1 // Verificar si existe el comando (input) en la tabla de comandos
2 int IsCommandInTable(const char *cmd) {
3     for (int i = 0; i < cmdCount; i++) {
4         if (strcmp(cmdTable[i], cmd) == 0) {
5             return 1;
6         }
7     }
8     return 0;
9 }
```

Función IsCommandInTable

```
1 // Liberar memoria de la tabla comandos
2 void FreeCommandsMemory() {
3     for (int i = 0; i < cmdCount; i++) {
4         free(cmdTable[i]);
5     }
6 }
```

Procedimiento FreeCommandsMemory

```
1 // Separa en Tokens el comando (input)
2 void TokenizeUserInput(char *command, char *tokens[]) {
3     char *token = strtok(command, " ");
4     int i = 0;
5     while (token != NULL) {
6         tokens[i++] = token;
7         token = strtok(NULL, " ");
8     }
9     tokens[i] = NULL;
10 }
```

Procedimiento TokenizeUserInput

```
1 // Imprime los tokens del comando (input)
2 void PrintTokens(char *tokens[]) {
3     for (int i = 0; tokens[i] != NULL; i++) {
4         printf("Token %d: %s\n", i, tokens[i]);
5     }
6 }
```

Procedimiento PrintTokens

```

1 // Calculo de Distancia Hamming
2 void DistanciaHamming(const char *str1, const char *str2, char *recommendations[], int *recommendationCount) {
3
4     if (strlen(str1) == strlen(str2)) { // Valida Longitud igual
5         int distancia = 0;
6
7         for (int i = 0; i < strlen(str1); i++) {
8             if (str1[i] != str2[i]) { // Revisar char por char si son diferentes
9                 distancia++;
10            }
11        }
12
13        // Agregar a recomendaciones
14        if (strlen(str1) <= 10) {
15            if (distancia <= (int)(strlen(str1) * 0.5)) {
16                // Guarda el comando si la distancia es menor o igual a la mitad de la longitud
17                recommendations[*recommendationCount] = strdup(str2);
18                (*recommendationCount)++;
19            }
20        } else if (strlen(str1) > 10) {
21            if (distancia <= (int)(strlen(str1) * 0.6)) {
22                // Guarda el comando si la distancia es menor o igual al 60% de la longitud
23                recommendations[*recommendationCount] = strdup(str2);
24                (*recommendationCount)++;
25            }
26        }
27    }
28 }

```

Procedimiento DistanciaHamming

```

1 // Comparar dos caracteres en quick sort
2 int Comparador(const void *a, const void *b) {
3     return (*(char *)a - *(char *)b);
4 }

```

Función Comparador

```

1 // Calculo de Anagramas
2 void SonAnagramas(const char *str1, const char *str2, char *recommendations[], int *recommendationCount) {
3
4     if (strlen(str1) == strlen(str2)) { // Valida Longitud igual
5
6         char *copia1 = strdup(str1);
7         char *copia2 = strdup(str2);
8
9         // Ordenar Los caracteres de Los strings
10        qsort(copia1, strlen(copia1), sizeof(char), Comparador);
11        qsort(copia2, strlen(copia2), sizeof(char), Comparador);
12
13        // Revisar igualdad de strings ordenados
14        if (strcmp(copia1, copia2) == 0) {
15            recommendations[*recommendationCount] = strdup(str2);
16            (*recommendationCount)++;
17        }
18
19        free(copia1);
20        free(copia2);
21    }
22 }

```

Procedimiento SonAnagramas

```

1 // Unir Los tokens de la recomendacion a un string
2 void JoinUserRecommendation(char *recommendation, char *tokens[], char *newCommand) {
3     strcpy(newCommand, recommendation);
4     strcat(newCommand, " ");
5
6     int i = 1;
7
8     while (tokens[i] != NULL) {
9         strcat(newCommand, tokens[i]);
10        strcat(newCommand, " ");
11        i++;
12    }
13
14    newCommand[strlen(newCommand) - 1] = '\0';
15 }

```

Procedimiento JoinUserRecommendation

```

1 // Imprimir la tabla de comandos
2 void ListCommandsTable() {
3     for (int i = 0; i < cmdCount; i++) {
4         printf("%s\n", cmdTable[i]);
5     }
6 }

```

Procedimiento ListCommandsTable

```

1 // Eliminar recomendaciones duplicadas
2 void DeleteDuplicatedRecommendations(char *recommendations[], int *recommendationCount) {
3     for (int i = 0; i < *recommendationCount; i++) {
4         for (int j = i + 1; j < *recommendationCount; j++) {
5             if (strcmp(recommendations[i], recommendations[j]) == 0) {
6                 free(recommendations[j]);
7
8                 for (int k = j; k < *recommendationCount - 1; k++) {
9                     recommendations[k] = recommendations[k + 1];
10                }
11                (*recommendationCount)--;
12                j--;
13            }
14        }
15    }
16 }

```

Procedimiento DeleteDuplicatedRecommendations

```

1  int main() {
2      printf("Bienvenido a dwimsh - Escrito por Daniel Juarez\n");
3
4      LoadCommands(); // Cargar comandos a memoria
5
6      char command[256];
7      char *tokens[256];
8      char *recommendations[MAX_CMDS];
9      int recommendationCount = 0;
10     pid_t pid;
11     int status;
12
13     do {
14         printf("dwimsh> ");
15         fgets(command, sizeof(command), stdin);
16
17         command[strcspn(command, "\n")] = 0;
18
19         TokenizeUserInput(command, tokens); // Tokenizar el comando
20
21         if (tokens[0] == NULL || strlen(tokens[0]) == 0) { // Si se ingresa un comando vacio
22             printf("No se ingreso ningun comando\n");
23             continue;
24         }
25
26         if (strcmp(tokens[0], "exit") == 0) { // Salir del programa
27             break;
28         } else {
29             if (IsCommandInTable(tokens[0])) { // Verificar si el comando está en la tabla de comandos
30                 pid = fork();
31                 if (pid < 0) {
32                     perror("Error en fork");
33                 } else if (pid == 0) {
34                     execvp(tokens[0], tokens);
35                     perror("Fallo en la ejecucion");
36                     exit(EXIT_FAILURE);
37                 } else {
38                     wait(&status);
39                 }
40             } else {
41                 printf("Comando no encontrado en /usr/bin\n");
42
43                 // Buscar recomendaciones
44                 for (int i = 0; i < cmdCount; i++) {
45                     SonAnagramas(tokens[0], cmdTable[i], recommendations, &recommendationCount);
46                     DistanciaHamming(tokens[0], cmdTable[i], recommendations, &recommendationCount);
47                 }
48
49                 if (recommendationCount == 0){
50                     printf("No se encontraron comandos similares. Intente de nuevo.\n");
51                 } else {
52                     DeleteDuplicatedReccomendations(recommendations, &recommendationCount); // Eliminar recomendaciones duplicadas
53                     printf("Recomendaciones Totales: %d\n", recommendationCount);
54                 }
55             }
56         }
57     } while (true);
58 }

```

Primera parte de la implementación del main


```

1
2     char userInput[3];
3     for (int i = 0; i < recommendationCount; i++) { // Imprimir recomendaciones
4         printf("Quiere decir: \"%s\", recommendations[i]);
5
6         for (int j = 1; tokens[j] != NULL; j++) {
7             printf(" %s", tokens[j]);
8         }
9
10        printf("\n? [s/n] ");
11        fgets(userInput, sizeof(userInput), stdin); // Leer confirmacion de recomendacion
12
13        userInput[strcspn(userInput, "\n")] = 0;
14
15        if (userInput[0] == 's' || userInput[0] == 'S') {
16            char newCommand[256];
17            JoinUserRecommendation(recommendations[i], tokens, newCommand); // Hacer string la recomendacion aceptada
18            char *newCommandTokens[256];
19            TokenizeUserInput(newCommand, newCommandTokens); // Tokenizar el nuevo comando
20
21            pid_t pid = fork();
22
23            if (pid < 0) {
24                perror("Error en fork");
25            } else if (pid == 0) {
26                execvp(newCommandTokens[0], newCommandTokens);
27                perror("Error al ejecutar el comando");
28                exit(EXIT_FAILURE);
29            } else {
30                int status;
31                wait(&status);
32            }
33            break;
34        } else if (userInput[0] == 'n' || userInput[0] == 'N') { // Mostrar siguiente recomendacion
35            continue;
36        } else {
37            printf("Ingrese una Opcion Valida.\n");
38            i--; // Repetir la recomendacion
39        }
40    }
41 }
42
43 // Vaciar recomendaciones
44 for (int i = 0; i < recommendationCount; i++) {
45     free(recommendations[i]);
46 }
47
48 recommendationCount = 0;
49 }
50 }
51 } while (1);
52
53 // Liberar memoria
54 FreeCommandsMemory();
55
56 return 0;
57 }

```

Segunda parte de la implementación del main