PA5 Report

File Systems

CS307 - Operating Systems

Mert Ekici 26772
ekicimert@sabanciuniv.edu

I. Introduction & Problem Description

In this programming assignment, some of the **file system** utilities and functions will be used as a correction mechanism for txt files. When a file is opened, there is a **file pointer** which moves accordingly with read and write operations on the file. Moreover, this file pointer can be modified by directly using **lseek** command and what will be done in this assignment is highly related to this file pointer.

For the Programming Assignment 5, a corrector program is designed which accepts the current working directory (the directory that contains this executable) as its **root** directory, and traverses all of its sub-tree (all the directories and files reachable from this root directory). During this process, it modifies (or corrects if necessary) files with a txt extension by checking the prefixes and surnames of the individuals who have a match from the database. It will be assumed that there is a simple **database.txt file inside the root directory** and it consists several information about some people (gender, name and surname). This table like database has an individual for each row where:

- the first column is for “gender”,
- the second column is for “name” and,
- the third column is for “surname” of this specific person.

f Esma Gonul
f Zeynep Sasmaz
m Enes Koyuncu
f Ayse Ozyilmaz
f Alev Yildirim
m mehmet Olca

Incorrectly written some_random_file.txt:

Ms. Enes karaman was on his way to the office. However there was too much traffic and this time he knew that **Ms. Ayse Akyilmaz** will not tolerate him arriving late to work, once again...

Corrected version of some_random_file.txt:

Mr. Enes Koyuncu was on his way to the office. However there was too much traffic and this time he knew that **Ms. Ayse Ozyilmaz** will not tolerate him arriving late to work, once again...

There are two possible prefixes in our case: “Mr.” and “Ms.”. They should be updated if they do not match with the gender of the person who has match on the database as can be seen from the example above. Additionally, last names also should be checked and updated accordingly. Before moving on to the required functions and implementation details there are some important notes and assumptions specified for PA5 as follows;

- Incorrectly written last names and prefixes will be the **same length** as the correct ones.
- During going back and forward with the fseek function, it can be assumed that prefix length is 3 (“Mr.” or “Ms.”) and there is only **one whitespace** between each word.
- Files should be opened in “r+” mode to prevent undesired situations.
- Location of the file pointer should be modified by fseek and and correct strings should be written with fputs. Creating another txt file is not allowed.
- Although it has a txt extension, “database.txt” under the root directory is the source for the database and it should not be updated. **Nevertheless, other “database.txt” files under sub-directories should be checked and corrected if necessary**.

II. Required Functions

There are some specific file system functions mandatory to use. These **required functions** are listed and explained below:

- ***fseek***

The direction and **the location of the file pointer can be changed man**ually by fseek function. Normaly when a file is being read, the file descriptor’s position also increases sequentially. However, once a name match is found in this assignment, **going backwards and forward must be done by the fseek** function which is a simple and direct wrapping of the lseek function we have seen in the lectures (only difference is that lseek is specific to UNIX whereas fseek works in any OS).

- ***fputs***

This function is useful to place or overwrite the text wherever the file pointer is at. To give an example let’s assume that the file pointer is at the location 8 and if we execute fputs(“some string”,fptr), starting from the 8^{th} character in the txt file, it wil overwrite the next 11 characters as “some string”. Keep in mind that fputs does not append **it overwrites on the existing text.**

"This is PA5 for CS307"
fputs("okay", fptr)
"Okay is PA5 for CS307"

- ***ftell***

This function is **not a must but might be useful** to use. It is used for the **debugging** purposes in my program inside the **printf** functions. As it could be understood from it's name, **ftell gives the current location of the file pointer** as a way to keep track of where the file descriptor is. It returns the location as long int.

III. Implementation

As a general summary of the program designed for PA5, it consists a node struct with 4 utility functions: create_DB(), print_DB(), find_DB() and the most importantly correction(). Node struct is defined for providing a linked-list property that will store the contents of "database.txt" which can be easily accessed during the directory traversal and correction process. In the function called create_DB, the information of each individual will be inserted into the linked-list as a node. Next, print_DB function is used for prompting the whole linked-list database (it is used for debugging and not necessary in the main function thus, this function is commented out). Inside the correction function, find_DB is used to check each word of the txt file that is being searched and returns the matching node if exists. Finally in general, the correction function recursively traverses the whole root directory and its sub-directories to check txt files and correct them using fseek and fputs.

struct node

```
struct node
{
    struct node *next;
    char gender;
    char *name;
    char *surname;
};
```

Node struct has 4 fields:

next holds the address of the next node
gender holds the character of gender as 'm' or 'f'
name holds the name of the individual as char array
surname holds the last name of the individual

The linked-list will be reached via the **global** variable struct node* head.

create_DB

In the database creation function, the root directory is opened using **opendir(".")** and all of its contents is checked iteratively to detect **"database.txt".** If the database file is found it is opened with fopen in **"r+"** mode as suggested, and each line is parsed using **fgets** and **strtok**. The first element of the line is stored as gender, the second one is stored as name and the third one is stored as surname in the newnode which will be inserted to the database linked-list. Additionally, insertion operations are done from the head side of the linked-list and the characters such as '\n', '\r' are escaped to avoid unwanted characters inside the strings.

print_DB

In the print function, head is passed to a curr node pointer and the whole linked-list traversed until a NULL pointer is reached. During the traversal, the contents of each node are printed. However, this function is commented out in the main functions since it is not needed to print anything during the correction. Basically, this functions has been used for **debugging** purposes.

```
void print_DB()
{
    struct node *curr = head;

    while (curr)
    {
        printf("Gender: %c ", curr->gender);
        printf("Name: %s ", curr->name);
        printf("Surname: %s\n", curr->surname);
        curr = curr->next;
    }
}
```

find_DB

In find_DB function, again the linked-list is traversed starting from the head similar to the printDB. Yet in this version; if one of the node's name matches with the entered name parameter, **the node with the match is returned. Otherwise, a NULL pointer is returned.** This function will be used **inside the correction function** to find matches and check if their prefixes and surnames are correctly written.

```
struct node *find_DB(char *name)
{
    struct node *curr = head;

    while (curr)
    {
        if (!strcmp(curr->name, name))
        {
            return curr;
        }
        curr = curr->next;
    }

    return NULL;
}
```

correction

The **main work** of this program is handled in this function: correction. The correction function receives two parameters which are **const char* dirName and int root**. Directory name is used as the input of **opendir** and **readdir** functions. Since the function correction is called **recursively** during the fix process, each correction call passes the name of each sub-directory. Furthermore, integer **root** variable is for letting the correction function know that the current directory **is the root directory or not**. If it is root (root == 1), the files with the name "database.txt" passed without doing anything. Otherwise (root == 0), they are checked just as the other txt files. In the recursive part of the correction function, pathname of the directory is formed but the directories with the name **"." and ".." are skipped** to avoid infinite loops and going outside of the root directory. If a txt file that fits the given description is found (considering database.txt case), then it is opened using **fopen** with **"+r" option** again but now file is being **read character by character** and if a **whitespace** or **newline** character is detected (word is found), **find_DB** function is called with the passed characters in the char array named **word**. If a match is found the essential part of the program happens which is explained sequentially in the following bullet points:

1. ***fseek(fp, -(5 + strlen(word)), SEEK_CUR); → Backwards - Prefix***
    - To check prefix, file pointer goes backward (5 plus the length of the name) from its current position. The number 5 consists 2 whitespaces and fixed three characters of the prefix ("Mr." or "Ms.").
    - Then, from the beginning of the prefix, the program reads the prefix char by char into pword character array. Based on the gender of the matched node (res: return value of the find_DB), it checks the prefix and updates if it is not correct.
    - Before the correction part, cursor again is moved backwards with fseek as 4 steps (length of the prefix plus one whitespace). After that, fputs is called with the correct prefix and fseek is called again to move one step forward. Now, file pointer points the first character of the name.
2. ***fseek(fp, (1 + strlen(word)), SEEK_CUR); → Forward - Surname***
    - To reach the beginning of the surname, fseek is moved forward by the size of the name plus 1 (whitespace again).
    - Then, from the beginning of the surname, the program reads the prefix char by char into nword character array. Using strcmp function nword and res->surname from the database are compared and if they are not the same, a modification is necessary.
    - Before correction part, fseek moves back the cursor by 1 plus strlen(nword) from the current position of the file pointer and calls fputs from there with the correct surname (res->surname). Finally, fseek is called again to go 1 more step forward and continue to search other words of the txt file.

After the necessary checks and updates on the detected word, word array is **reset** and correction function moves on with the other words. When the txt file is completed, corrector looks for other files or directories **until it scans all of the sub entities.** In the main function, create_DB is called first and corrector is called after (print_DB is used during tests and debugs but now it is commented out, its removed output can be seen from the figure below).