4- File Search

The ‘findlocation’ file is a bash script which is used to filter and print a specific location from a file. The bash script requires a six digits input that, in case that the input provided by the user is not valid (ex. User entered letters, or without 6 digits) the bash scripts prompt the user to enter an American prefix with 6 digits and then exit with code 1. After the user input the six digits the bash script calls the bash command grep (used to search patterns) and stores the result in a variable named line, if the command grep does not find the prefix entered by the user the variable line will be set as empty. If the variable is zero, then the system shall prompt the user that the prefix was not found and then exit the script with code 1. When succeed, the system shall call bash command sed (used to switch characters) to remove all the empty spaces and store the result on the line variable and then call sed again and remove the prefix and echo the result.

A screenshot of a computer

Description automatically generated with medium confidence

Image 4.1 shows an invalid input using letters.



Image 4.2 shows a valid input not found in the file.

Graphical user interface

Description automatically generated with medium confidence

Image 4.3 Shows a valid input and the expected result.

Constrains: Program is not allowed to use any level 3 function.

Similar to ‘findlocation’, ‘findlocationfast’ is used to filter and return specific data extracted from a file. ‘findfastlocation’ is a c file. The user shall run the code using the command ./findfastlocationfast [file] [prefix] in that order. Entering less arguments or entering an invalid prefix will throw an error and display to enter a valid North America prefix and exit the program with code 1. If the entered arguments are correct the system shall open the file in read only mode and store it in a variable called fd (file descriptor) otherwise the variable fd will have the value of -1 then procced to seek the text file and store the resulting location in a variable lseek. In case the system is not able to correctly seek the file it will throw an error message stating that an error occurred while seeking the file. If succeed to seek the file then system shall proceed to look if the file is correctly formatted to match the struct entry\_t, failing to verify the formatting is going to result in a formatting error and then exit the system returning 1 after closing the file. After verifying that the file is correctly formatted the system shall proceed to map the file into memory using mmap function and stores the result into a pointer variable ptr. If mmap function failed the value of ptr shall be MAP\_FAILED, this will result in an error mapping the file and the system returning 1 after closing the file.

The struct format is made to fit the row format of the document. The struct is composed of an array of 6 chars which hold the prefixes in the document (first 6 characters of the row), an array of 25 char that holds the name of the city and the state followed by spaces until filled, and a char by itself that represents a new line. The struct is represented in a data type of entry\_t.

Lookup\_time function allows us to iterate the file. Receives the ptr variable in entry\_t data type, the number of entries, and a pointer to the prefix that we entered. This function allow us to iterate the ptr in form of entry\_t called dict. Each iteration it will call the function compare entries that will substract the dict[m].number to the prefix that we entered. If the result of the substraction is 0, then means that we found the prefix on the file and we do not need to continue iterating. Function shall return the value of dict[m].place. If we iterate all the dict and we do not find an occurrence of the prefix then return null.

Lookup\_time result is stored in a variable named place. If the value of place is NULL the system shall display an error message stating that the prefix did not matched any prefix on the file. Otherwise, the system shall print the place after removing the with space. Format Place: [Place].

Lastly, the system needs to unmap the file by using the function munmap. If is able to unmap the file the system shall return with 0 code. Otherwise, the system shall print the corresponding error message and return error code 1.

* The use of the struct entry\_t allowed us to format the map in a way that let us iterate the map more efficiently.

|  |  |  |  |
| --- | --- | --- | --- |
| 585353 | Rochester NY | \n | Index n - 1 |
| 585354 | Rochester NY | \n | Index n |
| Number[6] | Place[25] | newline | Index n + 1 |

* Archive O(log(n)) was simple. Since the file was sorted, we were able to implement a binary search algorithm.

During the development of of findlocationfast we did not encountered major problems since the testing file that was provided to us was properly formatted to work with mmap and the rest of the functions. Our major problem was regarding using the level 3 functions such as fprintf that were later changed to write calls.

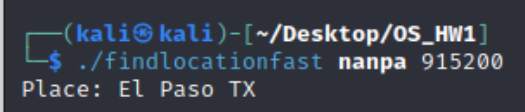


Image 4.4 Shows find location fast using a prefix present on the file.

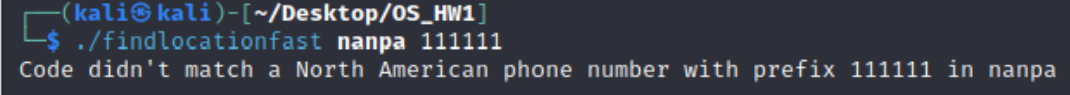


Image 4.5 Shows findlocationfast using an invalid prefix.

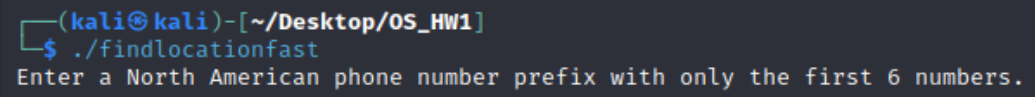


Image 4.6 Shows a call of findlocationfast with not enough arguments.

Text

Description automatically generated

Image 4.7 Shows a error message about having the nanpa file not properly formatted.