LOCAL SEARCH ENGINE

By:Team-3

Members

** Abhishek Upadhyay(Emp.ID=46192178)
** shivam Gupta(Emp.ID=46192287)
** Ravish Bilal(Emp.ID=46192182)
** Siddharth Vikas Kumar(EMP.ID=46192202)
** Priyam Jain (EMP.ID=46192205)
** Shivam Saraswat (EMP.ID=46192288)

Table of Contents

Intended Audience

Name	Roles
Suresh Kalidasan	Invigilator
Manjunatha Adinarayanappa	Mentor
Abhishek Upadhyay	Team Lead / Developer
Siddharth Vikas Kumar	Developer
Ravish Bilal	Developer
Sachin Gupta	Developer
Priyam Jain	Developer
Shivam Saraswat	Developer

Project Purpose

The purpose of the Local Search Engine is to help the user in searching for a particular file or string in the user's file system, the user can either search a file using any word or string or he can search using filename or just the extension or even with providing the absolute path.
It is designed to ease the users in finding and opening any particular file in the system whose whereabouts are unknown to the user.

Key Project Objectives

The key objective of the project is to provide the user with an interface that will help him/her in finding any particular file present in the user's system .
The project will display each and every occurrence of the file with its path in the form of a list and in case of a user wanting to search for a particular word or string in the entire system that he can opt for that option too.This application is compatible for all Unix systems.

Functional Overview

1) isFile(char * path)

Function to find whether the given path is a Directory or a file. This function returns an Integer (0-> directory, 1-> file,-1->if neither).

2) extractFileName(char *)

Function to extract name of file from file path

3) createFileNode()

Function to create a node in a Linked List to store search match information

4) searchInFile()

Open and Search file contents for search string.

5) OpenFile(char * fpath)

Function which takes the file path and prints its contents on the console

6) printLinkedList(File ** ptr)

Prints all file name and path stored Linked List

7) freeMemory(File ** ptr)

Free the dynamically allocated array of pointers and Linked List memory

8) allocateMemory()

Create an array of pointers and initialize to point to every node

9) searchLocalSystem(const char * path)

Search the system recursively

10) searchByWord()

Function to get the in file search string and decide search directory

11) searchByFilename()

Function to get the filename search string and decide search directory

12) openWithAbsolutePath()

Receive absolute file path from user and display file

Project Scope and Limitations

The given application will work fine on all Unix systems. The program should iterate all Directories and Subdirectories to search for a file and it should only provide user the reading accessibility to a file ,If read access is ot granted ignore the file or directory.

In Scope

- LSE provides an appropriate menu driven console user interface with options for the supported operations.
- LSE can search throughout the file system.
- We can also provide the base path for a file.
- It searches for a given word/sentence/string through all the files in the local file system and lists all the files where a match was found.
- LSE also provides the user an option to open the file whose contents the user wants to see.
- LSE can also search for a particular file when the absolute path of the file is provided and display the contents of that file.
- Local Search Engine can find all the files with specific extensions.

Out of scope

- Obtaining permission for read/access denied files/directories.
- Backtracking not supported once starting root node/path decided.
- LSE cannot search for Wildcard entries.

Limitations

1. Code is written using c and its platform dependent libraries so the executable file will not run on other operating systems like Windows, Mac, etc. For running in these OS, users have to create a new executable using ‘makefile’.
2. Wildcards processing not included in functionality, explicit path required.
3. LSE (Local Search Engine) will not work in the Unix root directory. It’ll start searching from the user's home directory (“/home/”).

Risks

Risk present in Our application.

1. We are using recursion to open directories and files so there can be a possibility that the stack can be full if there are a lot of files and directories present in our system, and that can cause our program to crash.

2. If there is a large file system , then it will take a lot of time for searching because we have created this project single process application.

Design Overview

The application interface begins by providing user 4 menu options namely:

1. Search By Word or String
2. Search By Filename
3. Open File With Absolute Path
4. Quit

1. Search By Word or String

This functionality is available to the user if the user opts for option 1 and this allows user to search a keyword or string in either a chosen location provided by the user or by default in the home directory of the user system,the program will display a list of all files containing that particular keyword or string

2. Search By Filename

This functionality is available to the user if the user opts for option 2 and this allows user to search a file by name in either a chosen location provided by the user or by default in the home directory of the user system,the program will display a list of all files with that name or containing that name as a substring.

3. Open File With Absolute Path

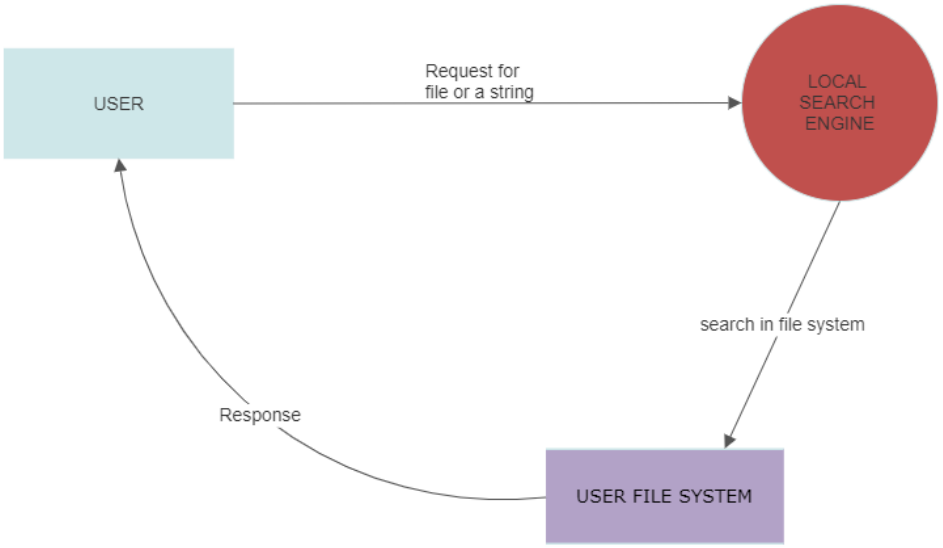
This functionality is available to the user if the user opts for option 3 and this allows the user to search a file at the user provided absolute path of the file .If the file is present at the location user can open the file only with read access.

4. Quit

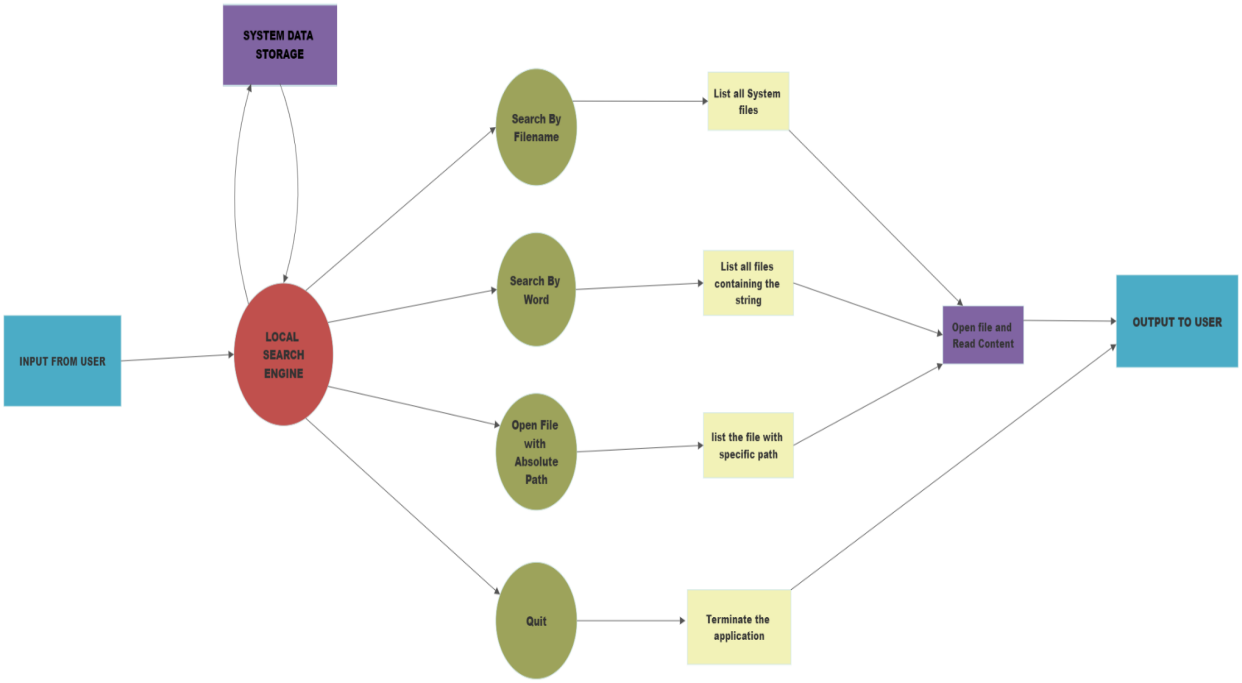
This functionality is available to the user if the user opts for option 4 and this allows the user to simply quit the interface.

If the user opts for any other option except these the program displays invalid choice and asks the user to choose again

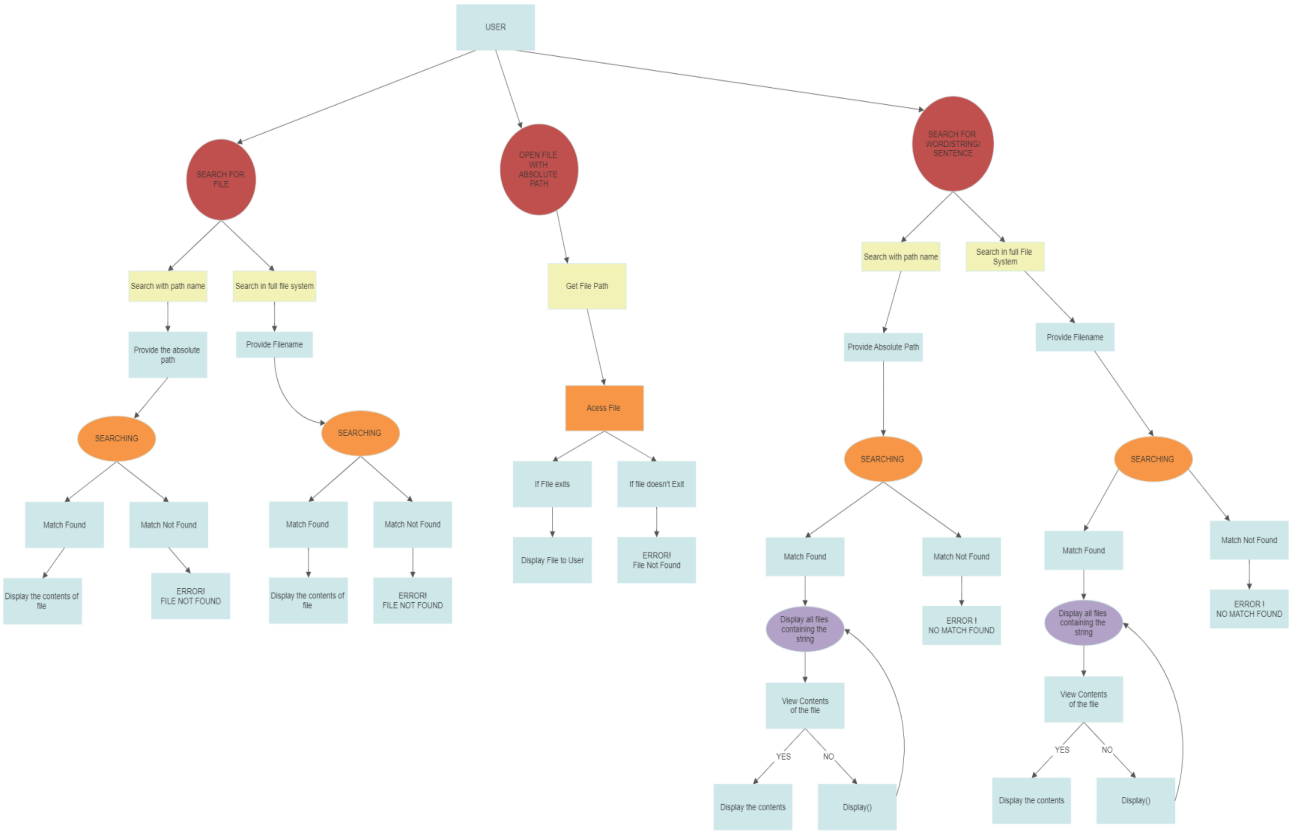
Level 0 diagram



Level 1 diagram



Data Flow Diagram



- **Error Detection and Error Handling**
  - **NULL Check**

    For every malloc and calloc used we have checked for the NULL pointer allocation
  - **Invalid Input Handling**

    For every input provided by the user is checked for validation against boundaries as well as invalidation
  - **File Extension Filter**

    Implemented Logic so that LSE can search for .txt,.c and .h files only which are in human readable format

Memory Management

Malloc is used to allocate memory for Linked List and array of pointers pointing to all node Locations. It is freed after the user decides not to open and view any more files that match their search parameters.
Memory is then reallocated using malloc call if the user decides to search a new string in the filesystem.
Since the path of a file can be a very long string we use calloc() to allocate memory and free it at the end of search.